
UNIT 1 DATABASE MANAGEMENT SYSTEM – AN INTRODUCTION

Structure

- 1.0 Introduction
- 1.1 Objectives
- 1.2 Need for a Database Management System
 - 1.2.1 The File Based System
 - 1.2.2 Limitations of File Based System
 - 1.2.3 The Database Approach
- 1.3 Logical DBMS Architecture
 - 1.3.1 Three Level Architecture of DBMS
 - 1.3.2 Mappings between Levels and Data Independence
 - 1.3.3 The Need of Three Level Architecture
- 1.4 Physical DBMS Structure
 - 1.4.1 DML Precompiler
 - 1.4.2 DDL Compiler
 - 1.4.3 File Manager
 - 1.4.4 Database Manager
 - 1.4.5 Query Processor
 - 1.4.6 Database Administrator
 - 1.4.7 Data files, indices and Data Dictionary
- 1.5 Database System Architectures
- 1.6 Data Models and Trends
- 1.7 Summary
- 1.8 Solutions/Answers

1.0 INTRODUCTION

In the present time, most of your online activities require interaction with a database system running as the backend of an application, such as purchasing from supermarkets or e-commerce website, depositing and/or withdrawing from a bank, booking hotel, airline or railway reservation, accessing a computerised library, ordering a magazine subscription from a publisher, using your smartphone apps to purchase goods. In all the above cases a database is accessed. Most of these backend database systems may be called Traditional Database Applications. In these types of databases, the information stored and accessed is textual or numeric. However, with advances in technology in the past decades, different newer database models have been developed, which will be discussed in Block 4 of this course. In this unit, we will be introducing the architecture and structure of a traditional Database Management System.

1.1 OBJECTIVES

After going through this unit, you should be able to:

- describe the File Based system and its limitations;
- describe the structure of DBMS;
- define the functions of DBA;
- explain the three-tier architecture of DBMS and its need; and
- appreciate different database models.

1.2 NEED FOR A DATABASE MANAGEMENT SYSTEM

A Database is an organised, persistent collection of data of an organisation. The database management system manages the database of an enterprise. Prior to use of the database systems, file-based systems were popularly used. In order to appreciate the strengths of database management systems, you may first list the problems associated with the file base systems, which are discussed next.

1.2.1 File Based System

Computerised file-based systems were primarily designed to make electronic versions of physical filing systems used by businesses. For example, a physical file can be set up to hold all the correspondence relating to a particular matter of a project, product, task, client or employee. In an organisation there could be many such files, which may be labeled and stored. Similarly, at homes you may have to maintain files relating to bank statements, loans, receipts, tax payments, etc. You can use electronic means to create these files. How do you search specific information from these files? For finding information, the entries could be searched sequentially. Alternatively, an indexing system could be used to locate the information directly, for example, you search for specific content in different files in your computer file search options.

The filing system works well when the number of items to be stored is small. It even works quite well when the number of items stored is quite large and they are only needed to be stored. However, a manual file system crashes when cross-referencing and processing of information in the files is carried out. For example, the University has several students who can register for different programmes of the University. The University has several faculty members who teach different courses to the enrolled students. The university may have to maintain separate files for the personal details of students, fees paid by them, and the details of the courses undertaken by them. In addition, the University also needs files for keeping details of each faculty member in various departments, courses taught by them. How would the following queries be answered?

- Fee paid by the students of the Computer Science Department per annum.
- The students who are willing to avail the pickup bus facility.
- Number of students taught by a faculty in a specific academic period.
- The number of students who have passed the programme this year in comparison to earlier years.
- How many students of a specific department have registered for courses other than their own department?

Please refer to *Figure 1*. The answer to all the questions cannot be computed by just simple statements but will require extensive file processing. Thus, each of these queries will require substantial amount of time in the file-based system.

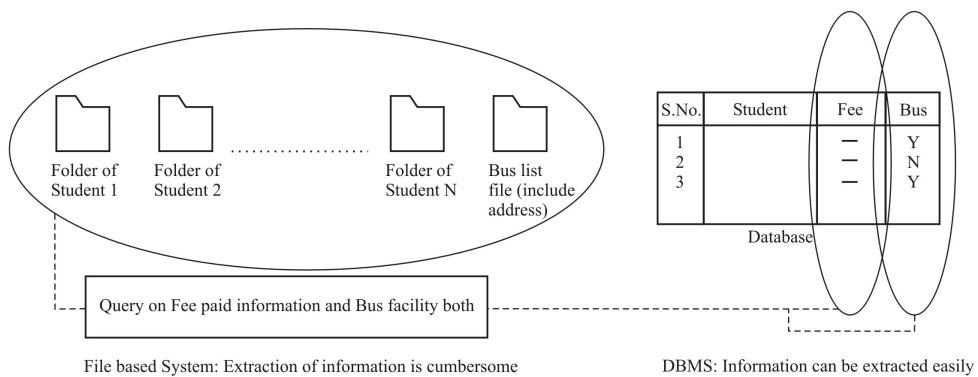


Figure 1: Information extraction using a file based system

1.2.2 Limitations of File Based System

File based systems required that several files should be opened for a particular system application. Several files may consist of duplicate data, which can result in several shortcomings. Some of these shortcomings are listed below:

- **Data isolation:** Since the file system stores data in separate files, which may belong to different applications. These files are not accessible to other applications and are difficult to share, especially when an application needs to use more than one file. Also, as the number of files can be very large for such systems, therefore, it would be difficult to search the relevant data from these files.
- **Data Duplication:** As stated earlier, a file system has different files for different applications, which may have overlapping data requirements. This will result in duplication of data, which can result in inconsistent data when duplicate data is updated. In addition, data duplication can also result in waste of storage. An example of data duplication is shown in Figure 1, where the address of several students may be stored in two different files, viz. student folder and bus list file.
- **Inconsistent Data:** The data in a file system can become inconsistent if more than one person modifies the data concurrently, for example, if any student changes the residence and the change is notified to only his/her file and not to the bus list. Entering wrong data is also another reason for inconsistencies.
- **Data dependence:** In the file systems, you need to clearly define the storage organisation of data files and the structure of the records in the application code. This means that it is extremely difficult to make changes to the existing structure, as any change in structure would require change in all the programs using that structure of the data. The programmer would have to identify all the affected programs, modify them and retest them. This characteristic of the File Based system is called **program data dependence**.
- **Incompatible File Formats:** Since the structure of the files is embedded in application programs, the structure is dependent on application programming languages. Hence the structure of a file generated by COBOL programming language may be quite different from a file generated by 'C' programming language. This incompatibility makes them difficult to process jointly. The application developer may have to develop software to convert the files to some common format for processing. However, this may be time consuming and expensive.
- **Fixed Queries:** File based systems are very much dependent on application programs. Any query or report needed by the organization would require the

application programmer to write a new program. As the type and number of queries or reports are expected to increase with time, producing different types of queries or reports would be difficult to implement in file-based systems. Since applications of file-based systems are designed to answer specific queries, therefore, new queries cannot be answered without generating an application. This entire process is time consuming and complex.

The file-based systems require large number of applications, therefore, are difficult to maintain. Further, each application would require separate provision from security. Besides the above, the maintenance of the File Based System is difficult and there is no provision for security. Further, data recovery from failures is inadequate or non-existent.

1.2.3 The Database Approach

As discussed in the previous section, the file system has many weaknesses. Therefore, a new approach was proposed that eliminates the weaknesses of the file system. This approach, called the database approach, separated data from application programs. The data in this approach is integrated from various applications and securely shared using a management system. *A database stores the integrated data of an organisation in a persistent manner.* The following are some of the characteristics of the database approach:

- The database can store data in a centralised database or a distributed database.
- The data is managed by a database management system (DBMS)
- It contains additional data about data, called metadata, which describes the structure and constraints on the stored data. The metadata is stored in DBMS in a data dictionary or system catalog.
- The database integrates the data of an organisation. This data is shared under the control of DBMS in a secure manner.
- DBMS allows several basic operations related to data, such as creating a database structure, inserting and editing data in a database, enforcing security and constraints on data and allowing access to data to authorised users.

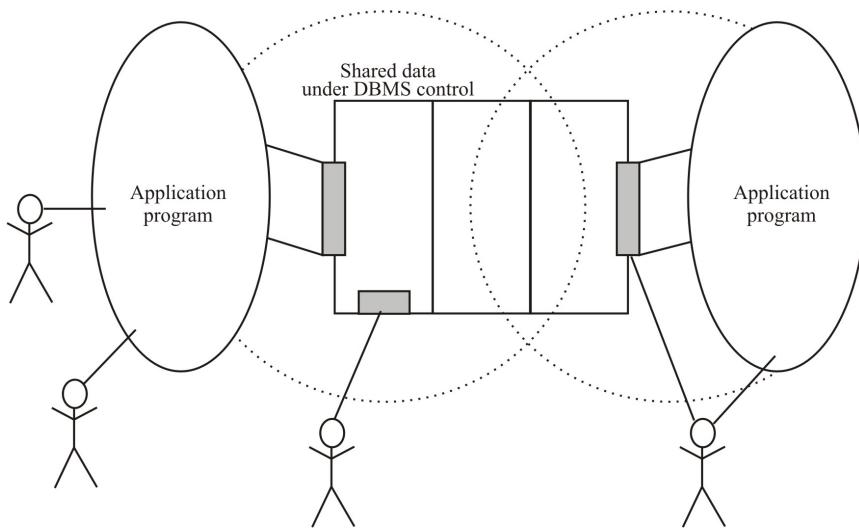
Let us discuss these advantages of database systems in more detail.

Reduction of Redundancies

The data of a database system is integrated, therefore, removes any duplicate data, as was the case of a file-based system, which maintains different files for different applications. The files that contain the copy of the data may become inconsistent as some of the files may be updated, whereas others may not be. The reduction in duplicate data may also help in reducing such data inconsistencies among the duplicated data of files. Thus, in the database approach, data is stored at a single place or with controlled redundancy under the control of DBMS, which helps in reducing data inconsistency.

Sharing of Data

The data of a database is integrated data of the entire organisation, however, the authorised users may be permitted to share partial data (why?). This scheme can be best explained with the help of a logical diagram (*Figure 2*). New applications can be built and added to the current system and data can be shared with these data, as per the need of such applications. Please note that data sharing is controlled by the DBMS, such that only the authorised users or applications can access it.



A user can either access data window through DBMS or use an application program.

Figure 2: User interaction to data through DBMS

Data Independence

In the file-based system, the descriptions of data and logic for accessing the data are built into each application program making the program more dependent on data. A change in the structure of data may require alterations to programs. Database Management systems separate data descriptions from data. Hence it is not affected by changes. This is called Data Independence, where details of data are not exposed. DBMS provides an abstract view and hides details. For example, in Figure 2, you can observe that the interface or window to data provided by DBMS to a user may still be the same although the internal structure of the data is changed.

Improved Integrity

Data Integrity refers to validity and consistency of data. Data Integrity means that the data should be accurate and consistent. This is implemented by enforcing checks or constraints on the data while it is being entered or manipulated in a database. Data of a database is not allowed to violate these constraints or rules. Constraints may apply to data items within a record or relationships between records. For example, a constraint on the age of an employee can be between 18 and 70 years only. While entering or modifying the data of the age of an employee, the DBMS should enforce this constraint. However, please note there is still a possible error if you enter the age of a person as 55 instead of 25. Such errors would require different ways of checking. Database systems support many other types of constraints, which will be discussed in later units.

Efficient Data Access

DBMS utilises techniques to store and retrieve the data efficiently, at least for unforeseen queries. An advanced DBMS allows its users to access data efficiently.

User Interfaces as per Users technical knowledge

A DBMS allows different types of interfaces, based on different levels of their technical knowledge. For example, the following types of interfaces may be provided by a DBMS:

- Menu driven forms and reports-based interfaces
- Application programming interface
- Query language interfaces
- Natural language interfaces

Representing relationship among data

Data of a database system is integrated data of an organization, which includes large number of related data objects. DBMS should maintain and preserve these relationships so that related data can be accessed easily.

Improved Security

The data of an organisation is vital and confidential. DBMS allows users to share only that information that they are authorised to access. For example, the critical information of an employee of an organisation should not be accessible to any other employee of that organisation. Hence, data of the database should be protected from unauthorised users. This is implemented by Database Administrator (DBA), who provides the users with controlled privileges on the data and type of operations. To enforce security, DBMS has a security and authorisation subsystem. Only authorised users may use the specific data of a database. Further, the operations performed by these users on data can be restricted to data retrieval, insertion, update, deletion etc. or any combination of these operations. For example, the Branch Manager of any company may have access to all data and is allowed to modify the incentive data of employees, whereas the Sales Assistant may not have access to salary details.

Improved Backup and Recovery

A file-based system may fail to provide measures to protect data from system failures, as taking backups periodically is the responsibility of the user. However, most of the DBMSs are designed in such a manner that it can recover from different types of hardware failures. In addition, DBMS also supports data backup. In general, a DBMS has a subsystem to support such provisions.

Support for concurrent Transactions

A transaction, in the context of a database system, is an atomic operation that is either completed fully or not at all. A database should allow multiple transactions to be carried out at the same time. For example, in a bank several money withdrawal and money transfer operations may be carried out at the same time. Most of the commercial DBMSs ensure that all these transactions do not interfere with each other.

Check Your Progress 1

- 1) What is a DBMS?

.....
.....

- 2) What are the advantages of a DBMS?

.....
.....

- 3) Compare and contrast the traditional File based system with Database approach.

.....
.....

1.3 THE LOGICAL DBMS ARCHITECTURE

As discussed in the previous section that most of the advantages of database systems are because of the creation of DBMS software. DBMSs must support many services and therefore are complex in nature. In addition, DBMSs are also required to store, manipulate and control a very large amount of data in a reliable manner. In this and subsequent section, we discuss the architecture of DBMS, which will help you with the processes and features of a DBMS.

In this section, two different architectures, which deal with two different aspects of database management, are discussed. The first architecture is the logical architecture,

which defines the data organisation and access at different logical levels of a database. The second architecture defines various components of a DBMS software. This architecture is referred to as physical database architecture.

1.3.1 Three Level Database Architecture

The three-level database architecture of a database defines the three different levels of abstraction of data for different types of users of the database. The proposed architecture was designed and standardised by the American National Standards Institute (ANSI) and is also known as ANSI/SPARC architecture. As per this architecture, a database schema can be visualised at three different levels. Figure 3 shows these three levels of this architecture. These levels are explained next.

The External or View Level

This level of abstraction provides a view of data for the users of a database system. Typically, this abstraction is created based on access rights of the users. Different types of users can be allowed different external views of data, as shown in Figure 3. Users can have different views of data. This level hides the overall structure of a database system from external users. From the database designer's point of view, this level also provides information on the mapping of external records of external views to the conceptual record of conceptual view of data.

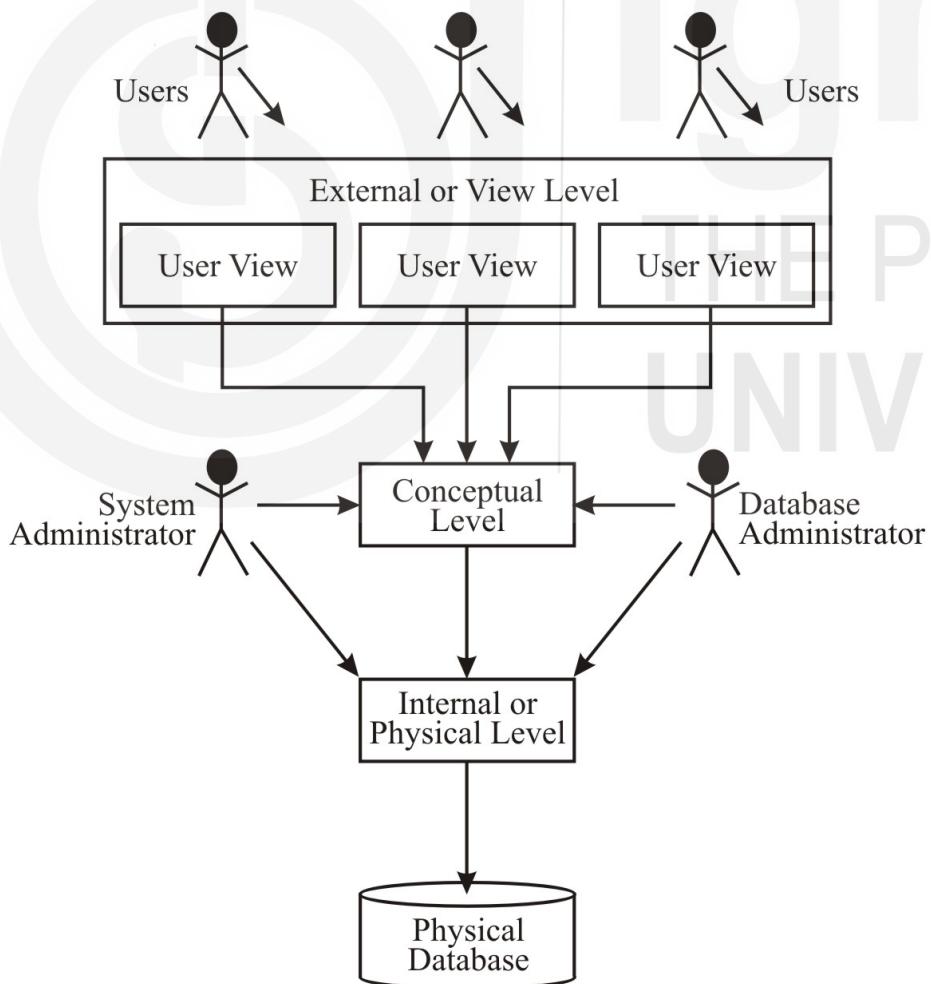


Figure 3: Logical DBMS Architecture

The Conceptual Level

The purpose of the conceptual level is to define the structure, relationships and constraints of a database system. Therefore, the conceptual level includes the structure of the data objects, relationships among those objects and the constraints on the data objects and access control of objects. A standard language called Data Definition Language (DDL) is often used in DBMSs to define the conceptual level or schema of a database system. The conceptual schema can be defined by the database administrator or the system administrator, as shown in Figure 3.

The Internal or Physical Level

A database system consists of many files, which include the data files, the meta data files, the access structure files, etc. The data files contain the actual data. The metadata files contain structure related information of data files, relationship details among data objects, constraints on data items etc. The access structure files define the control related information of the data objects. These files can be controlled by the DBMS software for access and updates. DDL can be used to describe the internal or physical schema too. In addition, this level can store additional files, which are used for enhancing the performance of the database system. These files are stored on an operating system, however, the access to these files is also under the control of the database system administrator.

1.3.2 Mappings between the three Levels and its relationship with Data Independence

The three level architecture defines a single database system across all the three levels. Therefore, different levels must map with each other. This mapping led to the concept of data independence, which was one of the major weaknesses in the file systems. This concept is explained next.

The first mapping is between the conceptual level and the external level. The external level is derived from the conceptual level. It is a part of the conceptual level, however, please note that these parts must be related else the database will lose database integrity. The advantage of this mapping is that an external user only needs to see the external level any change in the conceptual level will be hidden from the user. For example, at the conceptual level, you may keep information about the name of a person using the data items like *title*, *firstname* and *lastname*. At the external level, you may just map it to a data item *name* field. Thus, there exists a mapping, which will map:

name = *title* || *firstname* || *lastname* (|| is a concatenation operation)

Suppose, at a later point you decide to add additional data item *middlename* in the conceptual level, then you just need to change your mapping and not the programs which are based on the external level. The mapping would be changed to:

name = *title* || *firstname* || *middlename* || *lastname*

Thus, the conceptual to external mapping may isolate the external users from the changes in the conceptual level. These changes can be hidden in the conceptual to internal mapping. This is also referred to as logical data independence.

The second mapping is between the physical level and the conceptual level. This mapping insulates the conceptual level from changes in file organisation, indexes etc., without changing the conceptual level. In general, such changes may be performed to enhance the performance of a database management system. For example, instead of organising the student file on enrolment number, which is the primary key, you may organise the student file on Programme + Student name. This may enhance the access time of data for many important queries to the database. The conceptual level, in this case, still remains unchanged. This is also referred to as physical data independence.

1.3.3 Objective of the three-level architecture

The three level architecture separates the data that is presented to the user from the data that is stored in the database physically. The basic objectives of three level architecture are:

- It can support different views for different users. In case of change in any application, the views related to that application are required to change. Thus, three level architecture ensures the independence of data and programs.
- As stated above, due to independence of data and application programs, the applications need not deal with physical file organisation. Therefore, the application programs and users of a database are provided with a higher level of abstraction of data. Thus, a user or application programs are not required to deal with conceptual schema and the physical schema of a database. In general, the Database Administrator (DBA) is responsible for creating and modifying the conceptual schema and physical schema using DDL.

1.4 PHYSICAL DBMS ARCHITECTURE

A Database Management System (DBMS) is a complex software, as it manages many aspects of the database, such as query languages, data integrity, data security, access to files, and performance of data access. Figure 4 shows some of the important software components of a DBMS, which should be present logically in most of the DBMSs of today.

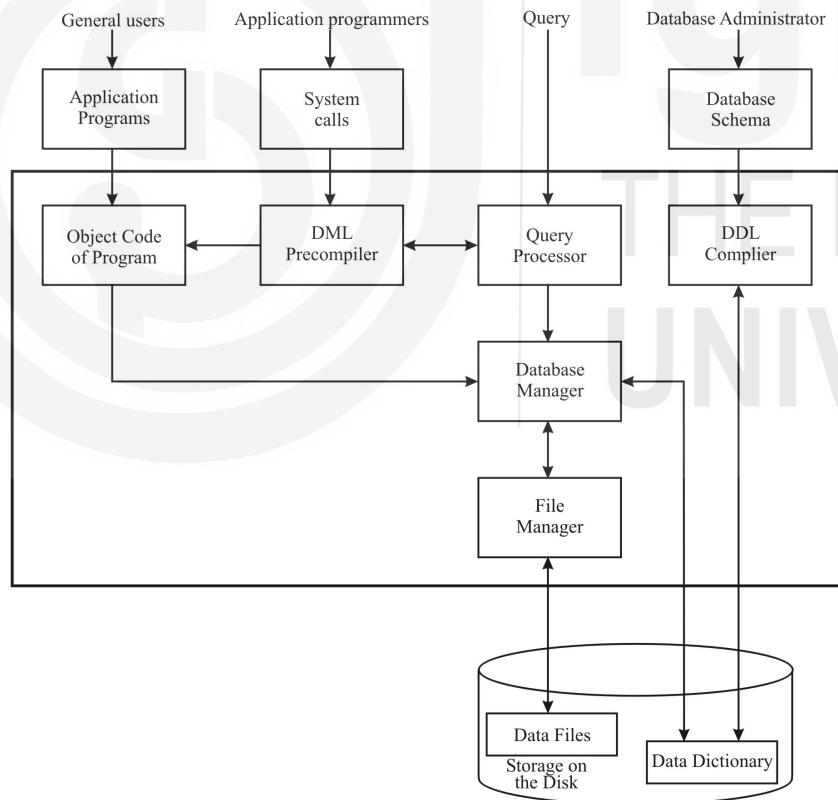


Figure 4: DBMS Structure

The software components of DBMS, as shown in Figure 4 are discussed next.

1.4.1 DML Precompiler

A DBMS consists of several languages, which are used for defining the data and for manipulation of data. For data definition, as stated earlier, a language called Data Definition Language (DDL) is used. Using DDL, you can define the structure of the data objects, data type of different data elements, integrity constraints, storage

constraints and access rights on the data. In addition, every DBMS consists of a Data Manipulation Language (DML), which is used for data insertion, modification, data access etc. The purpose of the DML precompiler is to translate the DML commands, which are written as part of application programs, into related procedure calls so that they can be executed to perform the desired operation. In addition, this component of the DBMS also coordinates the translation of queries, which are input to the query processor component.

1.4.2 DDL Compiler

A DDL compiler is used to compile a DDL statement into the related tables, which include the metadata, constraints, access rights etc. The metadata tables are stored in the data dictionary or system catalog. This metadata is used for providing information about the tables to other components of DBMS. In addition, the meta data is used for protecting data and enforcing data integrity.

1.4.3 File Manager

All the tables and metadata of the database are physically stored as files under the control of the file sub-system of the operating system. The file sub-system of the operating system has generic features, which may not be sufficient for a DBMS file. The File Manager component of the DBMS keeps track of all the data files, their index files and all other related information files of a database system. The final input/output to the database files are performed by the operating system.

1.4.4 Database Manager

Database Manager is one of the most important components of the DBMS. The role of the database manager is to accept the requests of data access or data manipulation by the user queries and application programs and perform the relevant action on the data taking the help of data dictionary and file manager. Database manager protects the data from unauthorised access and manipulation. It also enforces integrity constraints. You may please note that a database contains a large amount of data. The database manager is also responsible for deciding which data should be brought from secondary storage to main memory and back. Database manager allows simultaneous transactions on data and is also responsible for the recovery of the database in case of failures. It is also responsible for enhancing the performance of a database system access. The roles of a database manager are as follows:

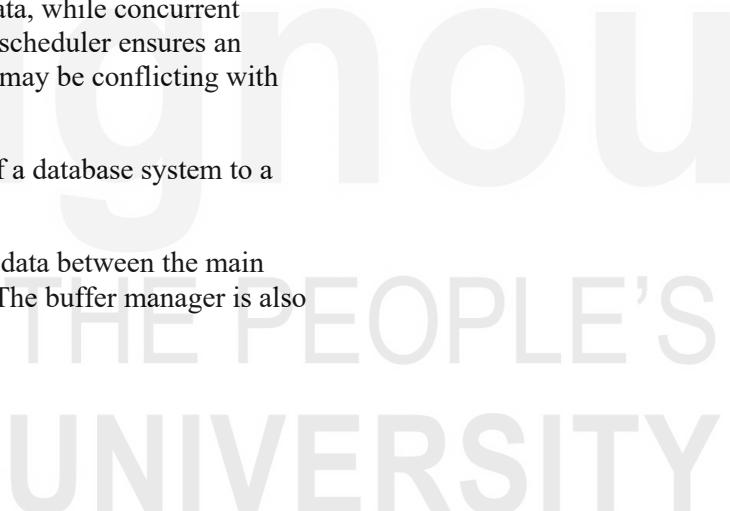
- **Collaborating with file manager for file handling:** As explained earlier, the file sub-system of the operating system is responsible for storing or manipulating data and related files of a database system. The file manager component of the DBMS interacts with the operating system for this purpose. The database manager collaborates and controls the file manager for various file operations.
- **Integrity enforcement:** Data integrity relates to correctness of data in a database system. A DBMS forces several constraints on data, which allows only correct data to be stored in a database system. These constraints are stored in the data dictionary and the database manager uses a data dictionary to enforce these integrity constraints. You will learn more about integrity constraints in the later units.
- **Security enforcement:** A DBMS does not allow unauthorised users to access the data of a database. This access control can relate to the entire data or a specific item of the data. For example, as an employee of an organization, you can read only your salary information. You do not have permission to change your salary. In addition, you cannot read or change the salary of any other employee. The security constraints can also be part of the data dictionary and implemented by the database manager.
- **Backup and recovery:** The purpose of the backup and recovery component of data manager is to guard the database against the loss of data at the time of a failure of the computer system. The failure of a computer system can occur due to many different reasons, such as hardware failure communication failure

software failure etc. The database manager ensures that data of various transactions is not corrupted even after a failure.

- **Concurrency control:** One of the major uses of database systems is in transaction management. In such systems, multiple transactions access and modify the database simultaneously. The database manager ensures that consistency of data is ensured even in the presence of concurrent transactions. Transaction management is discussed in block 3 in detail.

To perform the roles or functions as discussed above, the database manager has the following software components. These components are also shown in Figure 5.

- Authorisation control is used to verify the credentials of a user to ascertain if s/he is an authorised person to access or modify the data.
- Command Processor converts the DDL/DML or other programming language commands to an executable sequence of code.
- Integrity checker checks if the data that is being inserted in a database or is being modified follows all the specified constraints on the database.
- Query Optimizer tries to optimise the processing time of different queries.
- Transaction Manager is one of the important components, which checks if the user has the right to perform a transaction.
- Scheduler component manages the consistency of data, while concurrent transactions are being processed by a database. The scheduler ensures an ordering of transaction execution, even though they may be conflicting with each other.
- Recovery Manager is responsible for the recovery of a database system to a consistent state, in case of a failure.
- Buffer Manager optimises the process of transfer of data between the main memory and the disk, where the database is stored. The buffer manager is also termed as the *cache manager*.



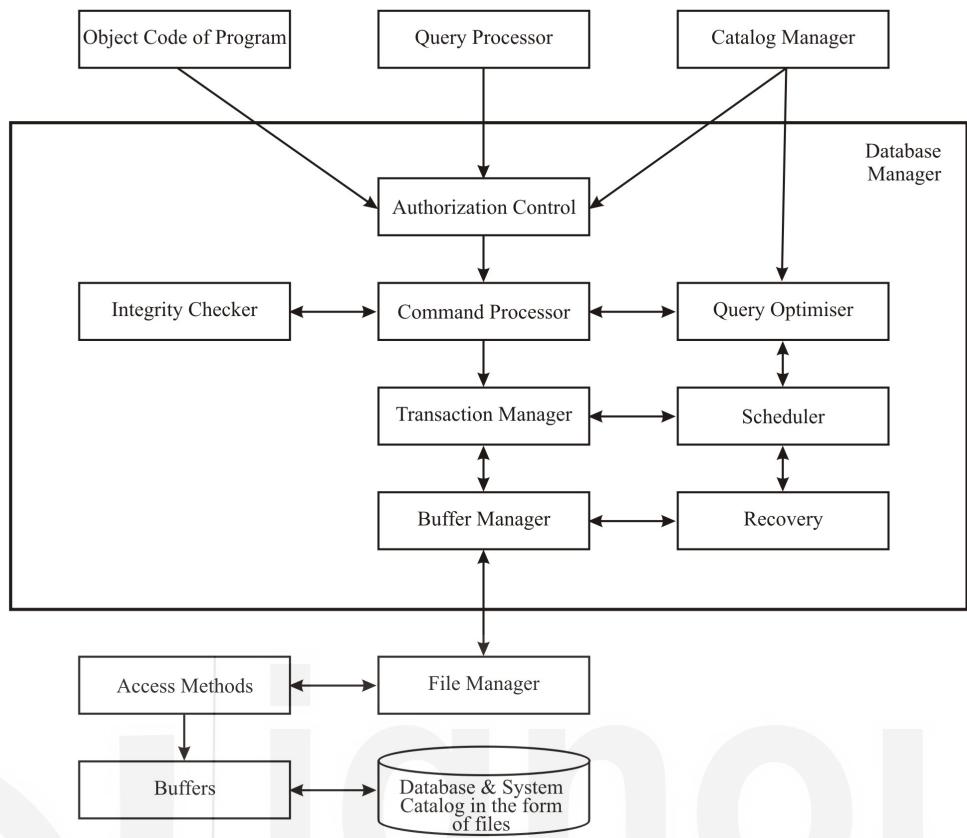


Figure 5: Components of Database Manager

1.4.5 Query Processor

Every DBMS consists of a query language which enables database creation, data manipulation, data retrieval and data control operations. This query language is usually a high level 4th generation language and is translated and optimised so that the required operations can be handled by the DBMS in the most optimal manner. In general, this query language processor consists of two basic components - the first one parses the query language into the native database language and the second optimises the query and prepares the code for query execution. The parser checks the syntax of the query, converts the query to a sequence of tasks and sends these tasks to the query optimiser. The query optimiser is responsible for generating a set of alternative query evaluation plans. It then estimates the expected total query response time, which includes access time of data from the disks, the time to execute a query and the time of communication of data or results over the network, for each of these query evaluation plans. Finally, the best query evaluation plan is selected to execute the query efficiently.

1.4.6 Database Administrator

The role of a database administrator (DBA) is to monitor the database creation (at different levels of 3-level architecture and mapping between the levels), modification, access, query response time, security, recovery in case of failure, etc. The functions of a DBA are given below:

- Defines the database Schema at different levels using DDL
- Specifies the organisation of the files and any other database access methods, including indexes, using DDL
- Performs the changes in the schema definitions, file organisation or indexes, whenever needed.
- Allows authorisation of data access or modification to different users.
- Specifies the integrity constraint on the database.

1.4.7 Data files, Indices and Data Dictionary

Basic Concepts

The data is stored in the data files. Data files can be standard files or encrypted files. These files, in general, are not accessible directly to any system user. The indices are stored in the index files. Indices provide fast access to data items. For example, a book database may be organised in the order of accession number of books yet may be indexed on Author name and Book titles for allowing faster access for searches on these attributes.

Data Dictionary: The data of an organisation is a valuable corporate resource. Commercial databases have large number of entities, attributes, data interrelationships and other information about the data. A data dictionary/directory is designed to store metadata, which is data about the data in a database system. A good data dictionary allows efficient data access, as it can provide a road map to guide users to access information within a large database. An ideal data dictionary may include the following information about a database:

- Schema definitions of internal, conceptual and external schemas, which includes:
 - the definition of every object or table
 - the attributes and their data types including primary key and attribute constraints.
 - any relationships between different attributes including foreign key constraints etc.
 - authorisation of access or modification at the level of attributes, if any.
 - Authorisations at the level of objects/tables, if any.
- Users and application programs of external schema and their permissions
- Database statistics such as number of records in every object/table, different number of occurrences of a data value in an attribute etc.

Check Your Progress 2

- 1) What are the major components of Database Manager?

.....
.....
.....

- 2) Explain the functions of the person who has control of both data and programs accessing that data.

.....
.....

1.5 DATABASE SYSTEM ARCHITECTURES

In the present time several database system architectures are popular.

First and earliest architecture of the database system was having a centralised database system. These systems contained a database in a single machine, which was either used by a single user or were shared by several users. A single user application in the present time may be an address book of your mobile device that is being used by a single user. However, today most of the centralised database systems use multiple core processors having large memory and multiple disks, called the database servers. A large number of users are connected to these servers, mostly through remote connections. These systems fall under the category of client-server systems. At its most basic level, this client server database architecture can be broken down into two parts: the *back end* and the *front end*.

The back end are the servers, which are responsible for managing the physical database and providing the necessary support and mappings for the internal, conceptual, and external levels. Other functions of a DBMS, such as security, integrity and access control, are also the responsibility of the back end.

The front end is just any application that runs on the DBMS. These may be applications provided by the DBMS vendor, the user, or a third party. The user interacts with the front end. A front-end may be user friendly interface provided by an application developed for database system access and modifications. These applications may be developed by the vendors themselves or by software developers using an Application Programming Interface (API) or third-party applications.

In this context, many different types of interfaces and utilities are offered in support of commercial database management systems. Some of these are:

- **Interfaces:**

- **Command line Interface:** This interface existed since the start of DBMS. You can use this interface to write DDL, DML and other permitted commands. These interfaces allow a very rich set of commands and are useful for expert users like DBA.
- **Graphical User Interface:** Such interfaces are developed to allow users to interact through database applications using menu driven interfaces. Such interfaces have become more useful over the last decade.
- **Utilities for Backup/Restore:** The purpose of these utilities is to ensure that the current state of a database is duly backed up on secondary storage, so that it can be recovered or restored in the case of any failure or even disaster. The backup is done periodically.
- **Utilities for Load/Unload of data:** Life of a database system is quite long. The size of a database system keeps increasing if the old data is not deleted. Also, hardware wears down over a period. Therefore, a database may be required to move from one machine to another machine. Sometimes the change in DBMS software or its versions also necessitates movement of data. The Load and Upload utilities are used for the purposes as above.
- **Utilities for Reporting or Analysis:** Reports form the important component of a database systems, especially management information systems. The reports and analysis of data can be used at various levels of decision making in an organisation. These utilities analyse and report data in various visual forms for decision making.

1.6 DATA MODELS AND TRENDS

A database system requires that the data should be structured in such a manner that it allows easier data access and manipulation operations. These structures are termed as database models. A database model has to address the following issues:

- It should define a logical structure of data, so that different attributes of data are easily identified.
- It should be able to represent relationships between different data elements or objects.
- It should support constraints on data elements and data objects.

The following Table defines some of the Data Models:

Model Type	Examples
Conceptual Model: Uses entities, their attributes and relationships among entities	<p><i>Entity-Relationship Model:</i> This model identifies the physical or logical entities and their attributes. In addition, this model also identifies the relationships among these entities. It is mainly represented in graphical form using E-R diagrams. This is very useful in Database design. The entity relationship diagram is discussed in this Block.</p>
Hierarchical Model: Uses data hierarchy	<p>Hierarchical Model: It defines data as and relationships through hierarchy of data values. <i>Figure 7</i> shows an example of hierarchical model. These models are now not used in commercial DBMS products.</p>
Record based Logical Models:	<p><i>Network Model:</i> Network model, as the name suggests, represents data about entities using a set of records. The relationships among these data records are represented using links. A simple network model example is explained in <i>Figure 6</i>. It shows a sample diagram for such a system. This model is a very good model as far as conceptual framework is concerned but is nowadays not used in database management systems.</p> <p><i>Relational Model:</i> It models data of both entities and relationships as tables. The relationship tables are related to entity tables using a set of constraints. This model is based on a sound mathematical theory of relations. This is one of the widely used data models and will be discussed in more detail in the subsequent units of this course.</p>
Object-based Models: Use objects as key data representation components.	<p><i>Object-Oriented Model:</i> An object in object-oriented model contains the data of an entity. In addition, the object also allows a set of defined operations on the data stored in an object. Further, the relationships among the objects are implemented by creating links among these objects and by implementing message passing. Object-Models are useful for databases where data interrelationships are complex, for example, Computer Assisted Design based components. These are explained in Block 4.</p>
Semi-structured Model: Uses user defined tags	<p><i>Semi-structured data Model:</i> Relational model is typically called a structured model, whereas, the semi-structured model uses user defined tags, which loosely force integrity contains. However, these models are very flexible in comparison to relational models. These models are popular in web-based data management, as they are very light database systems.</p>
Graph-based Models	<p><i>Graph data Model:</i> Graph is an important data structure consisting of nodes and arcs. The node can represent an entity while the links can be used to represent a relationship.</p>

Figure 6 shows the records of students and the result of the students in various courses taken by them. All the records of the students are shown as a single file. The student records can contain one or many pointers to the course file. Please notice in Figure 6 that the person named Ajay, whose enrolment number is 21026 has obtained 50 marks in course having course code 101. Further, he has obtained 19 marks in a course

having course code 204. Thus, each record on the course is linked to one record of the student.

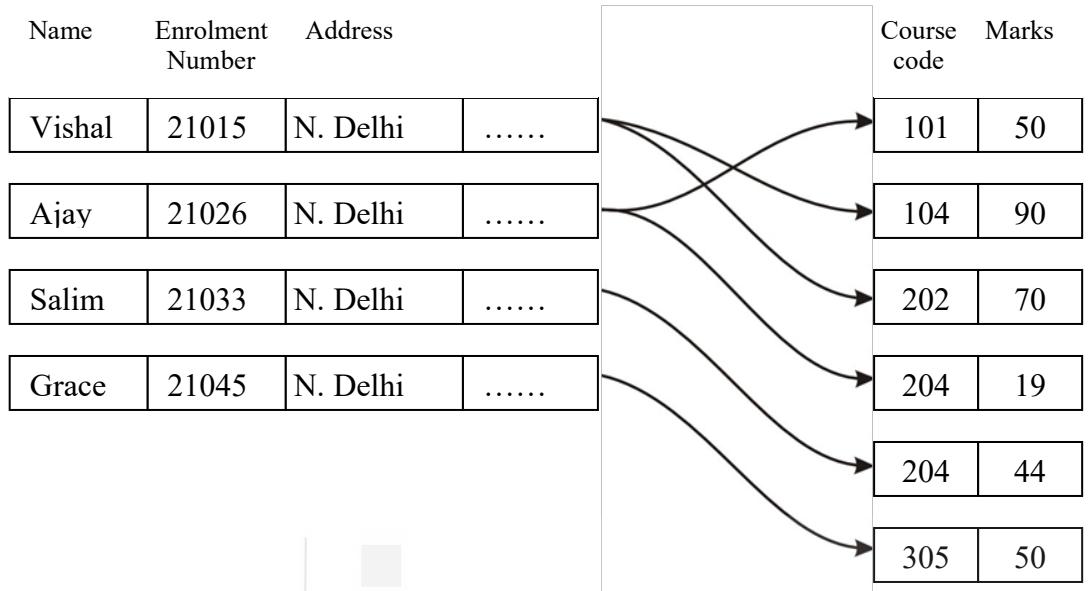


Figure 6: An example of Network Model

Figure 7 shows the data of Figure 6 in a hierarchical data model. Both these models are of historical nature. The relational model, object-oriented model and other model models are explained in the later blocks of this course.

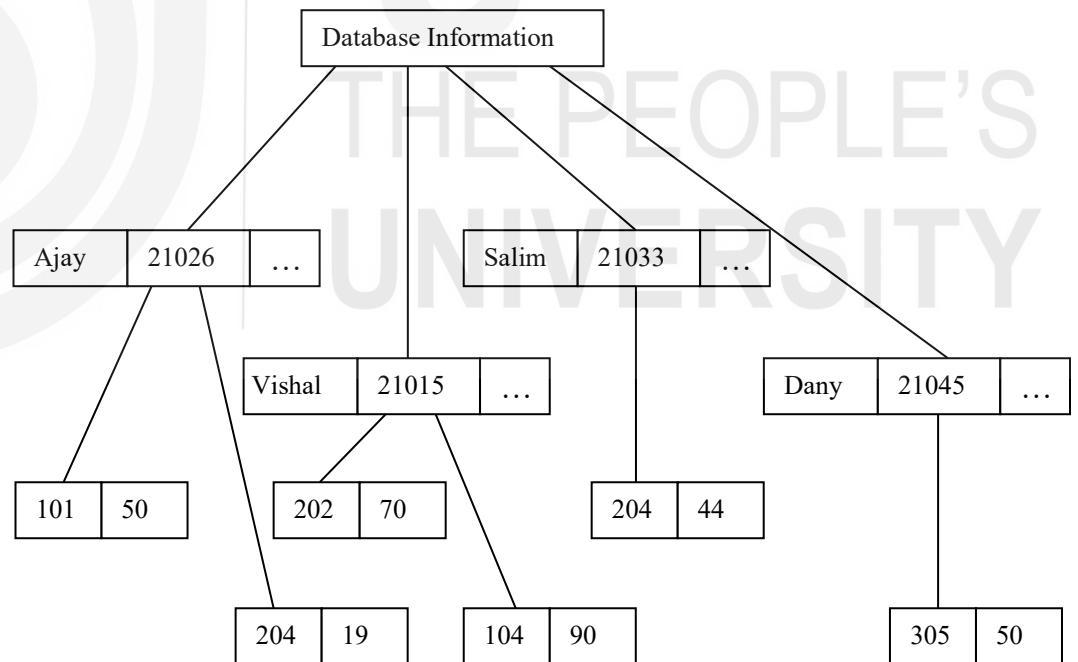


Figure 7: An example of Hierarchical Model

Relational Database management systems, which were supported by structured query language (SQL), became popular in the 1990s and later. These database systems were used to process transactions of a high-end web application. Such systems provided application users a graphical or menu driven interface using devices like computer, mobile devices and had features of user data input through keyboard, mouse, touch screens, forms, speech etc. With the growth of web applications lead to growth of semi-structured data. This led to use of eXtensible Markup Language (XML) and Java Script Object Notation (JSON) as the data exchange formats. Thus, several different database management systems, like geographical database management systems,

emerged that supported such data. With the rapid rate of growth of data newer database management systems like NoSQL databases started to grow. These databases lowered the time of application development but did not have strict control on consistency requirements. In the present time, the features of traditional and NoSQL database management systems are growing. Present day DBMSs are using cloud services for data storage and offer many database services at higher levels of abstraction leading to lesser time for database application deployment.

Check your Progress 3

- 1) What is a database backend?
.....
- 2) What is the need of utilities in a commercial database system? Explain any two such utilities.
.....
- 3) What is the difference between network and hierarchical models?
.....

1.7 SUMMARY

This unit provides you an introduction about the database system and database management system. It first defines the need of a database management system by comparing the database system approach to file system approach. The file system has several limitations due to data redundancy, however, the database approach integrates and shares the data among several applications. The unit also discusses the basic advantages of the database approach are - sharing of data, data independence, data integrity and security enforcement and transaction management for concurrent users.

The unit also explains the three-level architecture of database systems, which allows a database to be defined in terms of schemas at different levels for different types of users. Such a design allows access of relevant data to relevant users. In addition, the unit explains the physical architecture of DBMS and describes various components of DBMS. Next, the unit discusses the database system architecture for the commercial database and several data models. It also briefly presents the recent trends in DBMSs. You are advised to go through the further readings for more details on these topics.

1.8 SOLUTIONS/ANSWERS

Check Your Progress 1

- 1) DBMS manages the data of an organisation. It allows facilities for defining, updating and retrieving data of an organisation in a sharable, secure and reliable way.
- 2) The advantages of DBMS are:
 - Reduces redundancies
 - Provides environment for data independence
 - Enforces data integrity
 - Data Security
 - Answers unforeseen queries
 - Provides support for transactions, recovery etc.

3)

File Based System	Database Approach
Cheaper	Costly
Data dependent	Data independent
Data redundancy	Controlled data redundancy
Inconsistent Data	Consistent Data
Fixed Queries	Unforeseen queries can be answered

Check Your Progress 2

- 1) Integrity enforcement, control of file manager, security, backup, recovery, concurrency control, etc.
- 2) A database administrator is normally given such controls. His/her functions are: defining database, defining and optimising storage structures, and control of security, integrity and recovery.

Check Your Progress 3

1. Most of the database systems are deployed in a multi-user environment either as a centralised or distributed system. A database server, which is at the backend, manages and controls the physical data. It also provides integrity, security and access control to multiple client's requests.
2. Utilities are special purpose software, which are provided to support additional features in support of a DBMS. Two such utilities are for backup/restore – to protect data against any failure and load/unload – to load data from one hardware/software combination to a different hardware/software combination.
3. A network model stores basic entities as data records and points are used to represent the relationship between them. A hierarchical model uses almost similar model to a network model to represent data of entities but the relationship between them is implemented as a hierarchy.

UNIT 2 RELATIONAL DATABASE

Structure	Page Nos.
2.0 Introduction	
2.1 Objectives	
2.2 The Relational Model	
2.2.1 Domain, Attribute, Tuple and Relation	
2.2.2 Super keys Candidate keys and Primary keys for the Relations	
2.3 Relational Constraints	
2.3.1 Domain Constraint	
2.3.2 Key Constraint	
2.3.3 Integrity Constraint	
2.3.4 Update Operations and Dealing with Constraint Violations	
2.4 Relational Algebra	
2.4.1 Basic Set Operation	
2.4.2 Cartesian Product	
2.4.3 Relational Operations	
2.5 Summary	
2.6 Solution/Answers	

2.0 INTRODUCTION

In the first unit of this block, you have been provided with the details of the Database Management System, its advantages, structure, etc. This unit is an attempt to provide you information about relational model. The relational model is a widely used model for DBMS implementation. Most of the commercial DBMS products available in the industry are relational at the core.

The relational model is based on the theory of relations and was first proposed by E.F. Codd. In this unit we will discuss the terminology, operators and operations used in relational model.

2.1 OBJECTIVES

After going through this unit, you should be able to:

- describe relational model and its advantages;
- perform basic operations using relational algebra;
- identify the relational constraints in a database system;
- identify the key of a relation.

2.2 THE RELATIONAL MODEL

A model of a database system basically defines the *structure or organisation of data*, the *set of integrity constraints* on the data and a *set of operations* that can be performed on the data. The basic structure of data in a relational model is in the form of two-dimensional tables. A table in this model is called a relation. Such organisation of data simplifies enforcement of integrity constraints and database operations. The following table represents a simple relation:

PERSON_ID	NAME	AGE	ADDRESS
1	Sanjay Prasad	35	B-4,Modi Nagar, UP
2	Sharada Gupta	30	Pocket 2, Mayur Vihar, Delhi
3	Vibhu Datt	36	C-2, Saket, New Delhi

Figure 1: A Sample Person Relation

Following are some of the advantages of the relational model:

- **Ease of use:** The simple tabular representation of the database helps the user define and query the database conveniently. For example, you can easily find out the age of the person whose first name is “Vibhu”.
- **Flexibility:** Since the database is a collection of tables, new data can be added and deleted easily. Also, manipulation of data from various tables can be done easily using various basic operations. For example, you can add a telephone number field in the table in *Figure 1*.
- **Accuracy:** In relational databases the relational algebraic operations are used to manipulate data values in a database. These are mathematical operations and ensure accuracy (and less of ambiguity) as compared to other models. These operations are discussed in more detail in Section 2.4.

2.2.1 Tuple, Attribute, Domain and Relation

Before we discuss the relational model in more detail, let us first define some very basic terms used in this model.

Tuple: Each row in a table represents a record or information of a single object, which is also termed as a tuple. For example, Figure 1 has three records or tuples. A record/tuple consists of a number of attributes, which is defined next.

Attribute: A relation consists of a large number of columns. Each of these columns, which defines a separate data value, is termed as an attribute. The column name in a relation is generally related to the meaning of data items of that column. For example, *Figure 2* represents a relation PERSON. The columns PERSON_ID, NAME, AGE, ADDRESS and TELEPHONE are the attributes of the relation PERSON and each row in the relation represents a separate tuple (record).

Relation Name: PERSON

PERSON_ID	NAME	AGE	ADDRESS	TELEPHONE
1	Sanjay Prasad	35	B-4,Modi Nagar, UP	011-25347527
2	Sharada Gupta	30	Pocket 2, Mayur Vihar, Delhi	023-12245678
3	Vibhu Datt	36	C-2, Saket, New Delhi	033-1601138

Figure 2: An extended PERSON relation

The relation of Figure 2 consists of 5 attributes, therefore, each tuple in this relation is called a 5-tuple. Thus, if a relation has n attributes, then each record in that relation would be termed as *n*-tuple.

Domain: A domain is a set of permissible values that can be accepted by a specific attribute. For example, in Figure 2, if you assume that there may be a maximum of 100 persons in the relation, then you may assign PERSON_ID to a domain of integer values, which should be in the range from 1 to 100 only. Once you have assigned this domain, then you will not be able to assign any value below 1 and above 100 to PERSON_ID. The domain of attribute AGE can be integer values between 0 and 150. The domain can be defined by assigning a type or a format or a range to an attribute. For example, a domain for a number 501 to 999 can be specified by having a 3-digit number format having a range of values between 501 and 999. Domains need not be contiguous numbers. For example, the enrolment number of IGNOU has the last digit as the check digit, thus the enrolment numbers are non-continuous.

Relation: Each table is a relation. A relation is defined using two basic aspects, viz. schema and an instance.

Relational and E-R Models

Relational Schema: A relational schema denoted as R, specifies the relation's name, the list of attributes of the relation and the domain of each attribute, which is denoted as R (A₁, A₂, ..., A_n). The relation R has n attributes, which is also called the degree of a relation.

For example, the relation schema of the relation PERSON (see Figure 1) can be defined as the follows:

PERSON (PERSON_ID: Integer, NAME: Character, AGE: Integer,
ADDRESS: Character)

Since the PERSON relation contains four attributes, this relation is of degree 4.

Relation Instance or Relation State: A relation instance denoted as r is a collection of tuples for a given relational schema at a specific point of time.

A relation state r of the relation schema R (A₁, A₂, ..., A_n), also denoted by r(R) is a set of n-tuples

r = {t₁, t₂, ..., t_m} has m tuples
Where each n-tuple is an ordered list of n values
t = <v₁, v₂, ..., v_n>
where each v_i belongs to domain (A_i) or contains null values.

Let us elaborate the definitions above with the help of examples:

Example 1:

RELATION SCHEMA For STUDENT:

STUDENT (RollNo: string, name: string, login: string, age: integer)

RELATION INSTANCE

STUDENT				
	ROLLNO	NAME	LOGIN	AGE
t ₁	3467	Shikha	xyz@hotmail.com	21
t ₂	4677	Kanu	abc@gmail.com	20

Where t₁ = (3467, shikha, xyz@hotmail.com, 20) for this relation instance, the number of tuples m = 2 and each tuple contains n = 4 values.

Example 2:

RELATIONAL SCHEMA For PERSON:

PERSON (PERSON_ID: integer, NAME: string, AGE: integer, ADDRESS: string)

RELATION INSTANCE

In this instance, the number of tuples m = 3 and each tuple has n = 4 values.

PERSON_ID	NAME	AGE	ADDRESS
1	Sanjay Prasad	35	B-4, Modi Nagar, UP
2	Sharad Gupta	30	Pocket 2, Mayur Vihar, Delhi
3	Vibhu Datt	36	C-2, Saket, New Delhi

If a relation has proper integrity constraints, then each tuple of that relation would be valid tuple. It may be noted that ‘Null’ value can be assigned for some of the attributes where the values are unknown or missing. However, the Null’ value cannot be assigned to certain attributes, this will be explained in the later sections.

Ordering of tuples

In a relation, tuples are not inserted in any specific order. **Ordering of tuples is not defined as a part of a relation definition.** However, records may be organised later according to some attribute value in the storage system. For example, records in the PERSON table may be organised according to PERSON_ID. Such data organisation depends on the requirement of the underlying database application. However, for the purpose of display, you may get these tuples displayed in the sorted order of age. The following table is sorted by age. Please note that this is sorted relation is the same as that or relation displayed in the order of PERSON_ID as each and every tuple is the same. It is also worth mentioning that the relational model **does not allow duplicate tuples.**

PERSON

PERSON_ID	NAME	AGE	ADDRESS
2	Sharad Gupta	30	Pocket 2, Mayur Vihar, Delhi
1	Sanjay Prasad	35	B-4, Modi Nagar, UP
3	Vibhu Datt	36	C-2, Saket, New Delhi

2.2.2 Super Keys, Candidate Keys and Primary Keys for the Relations

As discussed in the previous section, ordering of relations does not matter and all tuples in a relation are unique. However, can you uniquely identify a tuple in a relation? To answer this question, let us discuss the concept of keys in relations.

Super Keys

A **super key** is an attribute or set of attributes used to identify the records uniquely in a relation.

For Example, in the Relation PERSON described earlier PERSON_ID is a super key since PERSON_ID is unique for each tuple/record. Similarly (PERSON_ID, AGE) and (PERSON_ID, NAME) are also super keys of the relation PERSON since their combination is also unique for each tuple/record.

Candidate keys

Super keys of a relation may contain extra attributes. A candidate key is a minimal super key, which means that a candidate key does not contain any extraneous attribute. An attribute is called extraneous if even after removing it from the key, the remaining attributes still has the properties of a super key. A relation may have several candidate keys. A candidate key may contain one or many attributes of a relation.

The following properties must be satisfied by a candidate key:

- In any instance of a relation, the value of the candidate key attribute(s) should be unique.
- You cannot put a ‘Null’ value in any attribute that is a part of the candidate key. This rule is also termed the entity integrity rule. Thus, a candidate key is unique and not null.
- A candidate key should have a minimal set of attributes.
- The value of a candidate key must be **stable**, which means it should not change frequently or its **value** change should not be outside the control of the system.

A relation can have more than one candidate key and one of them can be chosen as a **primary key**.

For example, in the relation PERSON, the two possible candidate keys are PERSON_ID and NAME (assuming unique names exist in the table). PERSON_ID may be chosen as the primary key.

Check Your Progress 1

Answer the following questions for the relational instance s of the following Supplier relation S:

S		
SNO	SNAME	CITY
S1	Smita	Delhi
S2	Jim	Pune

1. What are the different attributes of relation S and how many tuples does S have?
.....
2. What are the domains of the attributes of S?
.....
3. On sorting this relation on the field CITY, will the relation change? Will the order of tuples in the relation change?
.....
4. List the super keys, all the possible candidate keys, and the primary key of the relation.
.....

2.3 RELATIONAL CONSTRAINTS

A relational database is a collection of relations. Each relation consists of tuples, which consist of attributes. You can associate constraints, called relational constraints, with the attributes. Such constraints can be of the following types:

- Domain Constraints
- Key Constraint
- Integrity Constraints

2.3.1 Domain Constraint

These constraints bind the possible values of the attribute in a relation to a specific set of values, called the domain of the attribute. For example, an attribute COUNTRY may have a domain of names of all the possible names of the countries. In commercial relational database management systems, the data type associated with an attribute defines the broad domain of an attribute. These domains include:

- 1) Integral Data type like integer, long etc.
- 2) Data types related to real numbers like float, double etc.
- 3) Data types relating to alphabets like characters, strings etc.
- 4) Boolean

Thus, domain constraint specifies the possible set of values that you want to put in an attribute of a relation. The values that appear in each attribute/column must be drawn from the domain associated with that column.

For example, consider the relation:

STUDENT

ROLLNO	NAME	LOGIN	AGE
3467	Shikha	xyz@hotmail.com	21
4677	Kanu	abc@gmail.com	20

In the relation above, AGE of the relation STUDENT always belongs to the integer domain within a specified range (say 0 representing just born to 150) and not to strings or any other domain. Within a domain, non-atomic values should be avoided. This sometimes cannot be checked by domain constraints. For example, a database which has area code and phone numbers as two different fields will take phone numbers as-

Area_code	Phone
11	29534466

A non-atomic value in this case for a phone can be 1129534466, however, this value can be accepted by the Phone field only.

2.3.2 Key Constraint

This constraint states that the key attribute value in each tuple must be unique, i.e., no two tuples can contain the same value for the key attribute. This is because the value of the primary key is used to identify a unique tuple in a relation.

Example 3: If A is the key attribute in the following relation R, then A1, A2 and A3 must be unique.

R	
A	B
A1	B1
A3	B2
A2	B2

Example 4: In relation PERSON, PERSON_ID is the primary key so PERSON_ID cannot be assigned the same for two different persons.

2.3.3 Integrity Constraint

There are two types of integrity constraints:

- Entity Integrity Constraint
- Referential Integrity Constraint

Entity Integrity Constraint:

It states that any attribute of a **primary key cannot take a Null value**. This is because the primary key is used to identify individual tuple in the relation. You will not be able to identify the records uniquely if they contain null values for the primary key attributes. This constraint is specified for each relation of a database.

Example 5: Let R be a relation; an instance r of R is given below. Is the instance r valid?

A#	B	C
Null	B1	C1
A1	B2	C2
Null	B2	C3
A2	B3	C3
Null	B1	C5

Note: A ‘#’ in the headings row is indicating that A is the Primary key of R.

In the instance r of relation R above, the primary key has Null values in the tuple t_1 , tuple t_3 and tuple t_5 . As per the entity integrity constraint, Null value in primary key is not permitted. Thus, relation instance r is an invalid instance.

Referential integrity constraint

For defining the referential integrity constraint, first we explain the concept of a **foreign key** and **foreign key constraint**.

Consider an attribute set A of a relation R, which references an attribute set B in a relation S, then A will be referred to foreign key in R provided it fulfills the following conditions:

1. B is the Primary key of S.
2. A and B are defined over the same domains. A is called the foreign key in relation R, which references B in relation S.
3. For every value x of attribute set A, in any instance r of R, there exists a tuple in any instance s of S, where the value of attribute set B is x . Please note that there will be only one such tuple having the value x , as B is the Primary key of S. This is called foreign key constraint.
4. Please note that the relation R is a referencing relation and S is called referenced relation.

A referential integrity constraint is a more generic form of foreign key constraint, which requires that the attribute of the referencing relation must be present in the attribute of referenced relation in some tuple. Thus, foreign key constraint is a specific form of referential integrity constraint. Foreign key constraint is explained with the help of the following example.

Example 6: Consider the two relational instance r of relation R and s of relation S.

Now answer the following questions:

- a) If C in R is foreign key of C in S, then are the following instances r and s valid?
- b) A new tuple having values (A6, B2, C4) is added in r. Does it violate foreign key constraint?

Instance r of R

A#	B	C [^]
A1	B1	C1
A2	B2	C2
A3	B3	C3
A4	B4	C3
A5	B1	C5

Instance s of S

E	C#
E1	C1
E2	C3
E3	C5
E2	C2

Notes:

- ‘#’ identifies the Primary key of a relation.
- ‘^’ identifies the foreign key in a relation.

- a) In the example above, every value of C in r is matching with the values of C in s in some tuple. The r contains the values C1, C2, C3, C5 and s contains all these values. Thus, the instances r and s does not violate foreign key constraint.
- b) If you try to insert the tuple (A6, B2, C4) in r, then it is violating the foreign key constraint for the present instances of relations, as there is no tuple in s, which contains value C4 in s.

2.3.4 Data Updating Operations and Constraint Violations

There are three basic data updating operations to be performed on relations:

- Insertion of tuples
- Deletion of tuples
- Update the value of attribute(s) in a relation

The INSERT Operation:

- To add new records in a database system, you use insert operation. For example, in the instance r of R of example 6, you may like to add a new record <'A5', 25, 'C6'>. Addition of a new record may cause constraint violation, which are explained below:
- Violation of Domain constraint: Such a violation occurs if the domain of a value, which you are trying to insert in an attribute, does not match the attribute's domain. e inserting a numeric value 25 into attribute B, whose domain is alphanumeric sting. This will cause a domain constraint violation.
- Violation of Key constraint: This violation occurs when you try to insert a key value, which already exists in some tuple of existing relational instance. For example, when you are trying to insert tuple <'A5', 25, 'C6'> in R, you are inserting a value A5 in the key attribute A. However, the value A5 already exists in attribute A of tuple t₅. Therefore, it is a violation of key constraint.
- Violation of Entity Integrity constraint: This violation will occur; in case you try to insert a Null value into the primary key attribute. For example, if you try to insert a tuple <'Null', 25, 'C6'> in R, you are violating this constraint.
- Violation of Referential Integrity constraint: Consider that while inserting a new tuple, the value that you are trying to insert in a foreign key attribute, which refers to an attribute in another relation where it is the primary key, does not match with any value of referred attribute of the referenced relation. (Please see example 6 par (b)). For example, when you are trying to insert tuple <'A5', 25, 'C6'> in R, you are inserting a value C6 in the foreign key attribute A of R. However, the value C6 is not a value in referred attribute C in referenced relation S. Thus, insertion of this tuple violates the referential integrity constraint.

Dealing with constraints violation during insertion:

If the *insertion* violates one or more constraints, then two options are available:

- Default option: - *Insertion can be rejected and the reason for rejection can also be explained to the user by DBMS.*
- Ask the user to correct the data, resubmit, and give *the reason for rejecting the insertion.*

Example 7:

Consider the Relation PERSON of Example 2:

PERSON

PERSON_ID	NAME	AGE	ADDRESS
1	Sanjay Prasad	35	B-4, Modi Nagar, UP
2	Sharad Gupta	30	Pocket 2, Mayur Vihar, Delhi
3	Vibhu Datt	36	C-2, Saket, New Delhi

(1) Insert <1, 'Vipin', 20, 'Mayur Vihar'> into PERSON

Violated constraint: - Key constraint

Reason: - Primary key 1 already exists in PERSON.

How to resolve: - DBMS could ask the user to provide valid PERSON_ID value and accept the insertion if valid PERSON_ID value is provided.

(2) Insert <'null', 'Anurag', 25, 'Patparganj'> into PERSON

Violated constraint: - Entity Integrity constraint

Reason: - Primary key is 'null'.

How to resolve: - DBMS could ask the user to provide valid PERSON_ID value and accept the insertion if valid PERSON_ID value is provided.

(3) Insert <'abc', 'Suman', 25, 'IP college'> into PERSON

Violated constraint: - Domain constraint

Reason: - value of PERSON_ID is given a string which is not valid.

How to resolve: - DBMS could ask the user to provide valid PERSON_ID value and accept the insertion if valid PERSON_ID value is provided.

(4) Insert <10, 'Anu', 25, 'Patparganj'> into PERSON

Violated constraint: - None

Note: In the first 3 cases of constraint violations above DBMS will reject the insertion.

The Deletion Operation:

The delete operation is used to delete some existing records from a relation. To delete some specific records from the database a condition is specified based on which records are selected for deletion.

Constraints that can be violated during deletion

Only one type of constraints can be violated during deletion, it is referential integrity constraint. When you want to delete a record in a referenced relation, there is a possibility that you may delete a tuple whose attribute is being referenced by the referencing relation, where the attribute is the foreign key. Please go through example 8 very carefully.

Dealing with Constraints Violation

If the *deletion* violates the referential integrity constraint, then three options are available:

- Default option: - *Reject the deletion*. It is the job of the DBMS to explain to the user why the deletion was rejected.
- *Attempt to cascade (or propagate) the deletion* by deleting tuples whose foreign key references the tuple that is being deleted.
- Change the value of the *referencing attribute* that causes the violation.

Example 8:

Let instance r of relation R be:

A#	B	C^
A1	B1	C1
A2	B3	C3
A3	B4	C3
A4	B1	C5

And instance q of relation Q be:

C#	D
C1	D1
C3	D2
C5	D3

Note:

- 1) '#' identifies the Primary key of a relation.
 - 2) '^' identifies the Foreign key of a relation.
- (1) Delete a tuple with C# = 'C1' in Q.
 Violated constraint: - Referential Integrity constraint is violated
 Reason: - If the stated tuple is deleted from Q, the first tuple of R, which contains the Foreign key (C) value C1, will violate the foreign key constraint.
 How to resolve: Options available are
- 1) Reject the deletion.
 - 2) DBMS may automatically delete all tuples from relation Q and S with C # = 'C1'. This is called cascade deletion.
 - 3) The third option would result in putting NULL value in R where the value of attribute C is C1, which is the first tuple of R.

The Update Operations:

Update operations are used for modifying the values of attributes in a database. The constraint violations faced by this operation are logically the same as the problem faced by Insertion-Deletion Operation. You may define these actions yourself.

Check Your Progress 2

- 1 Consider the tables Suppliers, Parts, project and SPJ relation instances in the relations below.

SNO	SNAME	CITY
S1	Smita	Delhi
S2	Jim	Pune
S3	Ballav	Pune

PNO	PNAME	COLOUR	CITY
P1	Nut	Red	Delhi
P2	Bolt	Blue	Pune
P3	Pen	White	Mumbai

JNO	JNAME	CITY
J1	Sorter	Pune
J2	Display	Mumbai

SNO	PNO	JNO	QUANTITY
S1	P1	J1	200
S2	P2	J2	700

Using the instance of relations as given above, answer the following questions:

- 1 List a domain constraint for S.

.....

- 2 List the Primary keys of all the relations and primary key constraint for the SPJ relation.

.....

- 3 List all the Foreign keys and Foreign key constraints.

.....

- 4 What would be the constraint violations if the following operations are performed:

- (a) Insert <Null, 'Jack', 'Mumbai'> into S
- (b) Insert <'S2', Null, 'J3', 200> into SPJ
- (c) Insert <'P2', 'Pencil', 'Grey', 'Kolkata'> into P
- (d) Insert <'J3', 'Monitor', 'Jaipur'> into J

.....

.....

.....

2.4 RELATIONAL ALGEBRA

Relational Algebra is a set of basic operations used to manipulate the data in relational model. These operations enable the user to specify basic retrieval requests. The result of retrieval is a new temporary relation, which is computed after performing these operations on one or two relations. Some of these operations can be classified in three categories:

- Relational Operations
 - These can be unary operations
 1. SELECTION
 2. PROJECTION
 - Or binary operations
 1. CARTESIAN PRODUCT
 2. JOIN
- Basic Set Operations
 - 1. UNION
 - 2. INTERSECTION
 - 3. SET DIFFERENCE
- Assignment Operation, rename and other operations

In this section, we will discuss relational operations and basic set operations. You may refer to the other operations from the further readings.

2.4.1 Relational Operations

Relational algebra forms the basis of the query languages on a database system. Thus, by studying relational algebra, you may be able to write better queries for a database system. Let us discuss these operations in detail.

Selection Operation:

Selection is a unary operator, that is used to choose only those tuples from a relation that fulfill a given criteria. It is represented using the mathematical symbol σ , as given below:

You should specify a Boolean expression as a <Selection condition>. A selection condition uses comparison operators like { \leq , \geq , \neq , $=$, $<$, $<$ } and logical connectors like and (\wedge), or (\vee) and not (\neg).

Example 13:

Consider the relation PERSON. If you want to display details of persons having age less than or equal to 30, then the selection operation will be used as follows:

$\sigma_{\text{AGE} \leq 30} (\text{PERSON})$

The resultant relation will be as follows:

PERSON_ID	NAME	AGE	ADDRESS
2	Sharad Gupta	30	Pocket 2, Mayur Vihar, Delhi

Note:

- 1) Selection operation is commutative, i.e.,

$\sigma_{<\text{condition1}>} (\sigma_{<\text{condition2}>} (R)) = \sigma_{<\text{condition2}>} (\sigma_{<\text{condition1}>} (R))$

Hence, you can apply a sequence of Selection operations in any order

- 2) You can apply more than one condition by using logical connectors.

Projection operation

The projection operation is used to select specific attributes from amongst all the attributes of records. This is denoted as Π .

$\Pi_{\text{List of attributes for projection}} (\text{Relation})$

Example 14:

Consider the relation PERSON. If you want to display only the names of persons, then the projection operation will be used as follows:

$\Pi_{\text{NAME}} (\text{PERSON})$

The resultant relation will be as follows:

NAME
Sanjay Prasad
Sharad Gupta
Vibhu Datt

Please note that for projection operation:

$\Pi_{<\text{List1}>} (\Pi_{<\text{list2}>} (R)) = \Pi_{<\text{list1}>} (R)$ provided $<\text{list2}>$ contains attributes in $<\text{list1}>$.

2.4.2 Cartesian Product

Given two relations R and S, the cartesian product of these two relations can be represented as $T = R \times S$. The cartesian product operation produces all possible combinations of the tuples of tuples of relations R and S. The cartesian product operation is commutative as well as associative. In addition, the degree of the resultant

relation (T) is the sum of the degrees of relation R and relation S. The Cartesian product can be expressed using the following mathematical expression:

$T = \{t_1 \parallel t_2 \mid t_1 \in R1 \wedge t_2 \in R2\}$. Here, \parallel is a concatenation operator.

Example 12: Consider the following two relational instances of relation R1 and R2. What would be the cartesian product of the relations?

R1

A	B
A1	B1
A1	B2
A2	B2
A2	B3

R2

C
C1
C2

The Cartesian produce of the relations is denoted by $R3 = R1 \times R2$. The relation R3 will be as shown below:

A	B	C
A1	B1	C1
A1	B1	C2
A1	B2	C1
A1	B2	C2
A2	B2	C1
A2	B2	C2
A2	B3	C1
A2	B3	C2

Please note that in cartesian product every tuple/record of R1 is concatenated with every record of R2. You can observe that the record $\langle A1, B1 \rangle$ is concatenated with both the records of R2, that is why in the output R3 you have tuples $\langle A1, B1, C1 \rangle$ and $\langle A1, B1, C2 \rangle$, likewise for all the tuples of R1. Please also note that R1 has 4 tuples and R2 has 2 tuples, therefore, R3 has $4 \times 2 = 8$ tuples.

The JOIN operation

JOIN is a binary operator. It combines two relations based on a given join condition.

A JOIN operation is represented by mathematical symbol \bowtie . But how does JOIN operation combine two relations? It would require that there should be at least one attribute in each of the relations, which have the same domain. Such attributes are called domain compatible attributes.

Syntax:

$R1 \bowtie_{\text{join condition}} R2$ is used to combine related tuples from two relations R1 and R2 into a single tuple.

join condition is of the form:
 $\langle \text{condition} \rangle \text{AND} \langle \text{condition} \rangle \text{AND} \dots \text{AND} \langle \text{condition} \rangle$.

- Degree of Relation:
 $\text{Degree}(R1 \bowtie_{\text{join condition}} R2) \leq \text{Degree}(R1) + \text{Degree}(R2)$.
- Three types of joins are there:
 - Theta (θ) join**

When each condition is of the form A θ B, where A is an attribute of R1 and B is an attribute of R2; both A and B have the same domain; and θ is one of the comparison operators { \leq , \geq , \neq , $=$, $<$, $>$ }.

b) **Equijoin**

Equijoin is a restricted form of When each condition appears with equality operator ($=$) only.

c) **Natural join**

Natural join is defined as a specialised type of join, when the joining attributes in the two joining relations have the identical name, in addition to the identical domain and the condition for the join operation is equality ($=$) condition. In such cases only one joining attribute is kept in the result.

The following example explains the Natural join operation.

Example 15:

Consider the instance of the following relations. The primary key of STUDENT relation is ROLLNO and foreign key is COURSE_ID, which references the primary key COURSE_ID of COURSE relation.

STUDENT

ROLLNO	NAME	ADDRESS	COURSE_ID
100	Kanupriya	234, Saraswati Vihar.	CS1
101	Rajni Bala	120, Vasant Kunj	CS2
102	Arpita Gupta	75, SRM Apartments.	CS4

COURSE

COURSE_ID	COURSE_NAME	DURATION
CS1	MCA	3yrs
CS2	BCA	3yrs
CS3	M.Sc.	2yrs
CS4	B.Sc.	3yrs
CS5	MBA	2yrs

Display the name and other details of all the students along with their course details.

Solution: You may observe that the course details are existing in the COURSE relation and student names and other details are in the STUDENT relation. Therefore, you need to join these two relations to extract information. The join attributes in this case are COURSE_ID in STUDENT and COURSE_ID in the COURSE relation and equality operator is to be used; therefore, you use natural join operation, which can be represented as:

$$\text{STUDENTCOURSE} = \text{STUDENT} \bowtie \text{COURSE}$$

The output of this natural join will be the following relation. Please note that the output has just one COURSE_ID attribute.

STUDENTCOURSE

ROLLNO	NAME	ADDRESS	COURSE_ID	COURSE_NAME	DURATION
100	Kanupriya	234, Saraswati Vihar.	CS1	MCA	3yrs
101	Rajni Bala	120, Vasant Kunj	CS2	BCA	3yrs
102	Arpita Gupta	75, SRM Apartments.	CS4	B.Sc.	3yrs

There are other types of joins like outer joins. You must refer to further reading for more details on those operations. They are also explained in later blocks of this course.

2.4.3 Basic Set Operation

The set operations are the binary operations, i.e., each is applied to two sets or relations, which should be union compatible. Two relations R (A_1, A_2, \dots, A_n) and S (B_1, B_2, \dots, B_n) are said to be union compatible if they have the *same degree n* and domains of the corresponding attributes are also the same, i.e., $\text{Domain}(A_i) = \text{Domain}(B_i)$ for $1 \leq i \leq n$.

Union

If R1 and R2 are two union compatible relations, then $R3 = R1 \cup R2$ is the relation containing tuples that are either in R1 or in R2 or in both. In other words, R3 will have tuples such that $R3 = \{t \mid t \in R1 \vee t \in R2\}$.

Example 9:

R1

A	B
A1	B1
A2	B2
A3	B3
A4	B4

R2

X	Y
A1	B1
A7	B7
A2	B2
A4	B4

R3

$R3 = R1 \cup R2$ is

A	B
A1	B1
A2	B2
A3	B3
A4	B4
A7	B7

Note: 1) Union is a commutative operation, i.e,

$$R \cup S = S \cup R.$$

2) Union is an associative operation, i.e.

$$R \cup (S \cup T) = (R \cup S) \cup T.$$

Intersection

If R1 and R2 are two union compatible functions or relations, then the result of $R3 = R1 \cap R2$ is the relation that includes all tuples that are in both the relations

In other words, R3 will have tuples such that $R3 = \{t \mid t \in R1 \wedge t \in R2\}$.

Example 10:

R1

A	B
A1	B1
A2	B2
A3	B3
A4	B4

R2

X	Y
A1	B1
A7	B7
A2	B2
A4	B4

$R3 = R1 \cap R2$ is

A	B
A1	B1
A2	B2
A4	B4

- Note: 1) Intersection is a commutative operation, i.e.,
 $R1 \cap R2 = R2 \cap R1$.
 2) Intersection is an associative operation, i.e.,
 $R1 \cap (R2 \cap R3) = (R1 \cap R2) \cap R3$

Set Difference

If $R1$ and $R2$ are two union compatible relations, then the result of $R3 = R1 - R2$ is the relation that includes only those tuples that are in $R1$ but not in $R2$. In other words, $R3$ will have tuples such that $R3 = \{t \mid t \in R1 \wedge t \notin R2\}$.

Example 11:

R1

X	Y
A1	B1
A7	B7
A2	B2
A4	B4

R2

A	B
A1	B1
A2	B2
A3	B3
A4	B4

$R1 - R2 =$

A	B
A7	B7

$R2 - R1 =$

A	B
A3	B3

- Note: -1) Difference operation is not commutative, i.e.,
 $R1 - R2 \neq R2 - R1$
 2) Difference operation is not associative, i. e.,
 $R1 - (R2 - R3) \neq (R1 - R2) - R3$

Please note that a relational query language would be relationally complete, if it supports five relational algebraic operators – Selection, Projection, Union, Set Difference and Cartesian Product.

Check Your Progress 1

- 1) A database system is fully relational if it supports a language as powerful as _____.
- 2) Primitive operations are union, difference, product, selection and projection. The $A \cap B$ can be computed using
- 3) Which of the following is not a traditional set operator in relational algebra?

- a. Union
 b. Intersection
 c. Difference
 d. Join
- 4) Consider the relational instances of the relations Suppliers, Parts, project and SPJ relations given below. (Underline represents a key attribute. The SPJ relation has three Foreign keys: SNO, PNO and JNO.)

S	SNO	SNAME	CITY
S1	Smita	Delhi	
S2	Jim	Pune	
S3	Ballav	Pune	
S4	Seema	Delhi	
S5	Salim	Agra	

P	PNO	PNAME	COLOUR	CITY
P1	Nut	Red	Delhi	
P2	Bolt	Blue	Pune	
P3	Part1	White	Mumbai	
P4	Part2	Blue	Delhi	
P5	Camera	Brown	Pune	
P6	Part3	Grey	Delhi	

J	JNO	JNAME	CITY
J1	Sorter	Pune	
J2	Display	Bombay	
J3	OCR	Agra	
J4	Console	Agra	
J5	RAID	Delhi	
J6	EDP	Udaipur	
J7	Tape	Delhi	

SPJ	SNO	PNO	JNO	QUANTITY
S1	P1	J1	200	
S1	P1	J4	700	
S2	P3	J2	400	
S2	P2	J7	200	
S2	P3	J3	500	
S3	P3	J5	400	
S3	P4	J3	500	
S3	P5	J3	600	
S3	P6	J4	800	
S4	P6	J2	900	
S4	P6	J1	100	
S4	P5	J7	200	
S5	P5	J5	300	
S5	P4	J6	400	

Using the in the relations above, which of the following operations and constraints would be valid:

- i. UPDATE Project J7 in J setting CITY to Nagpur.
 - ii. UPDATE part P5 in P, setting PNO to P4.
 - iii. UPDATE supplier S5 in S, setting SNO to S8, if the relevant update rule is RESTRICT.
 - iv. DELETE supplier S3 in S, if the relevant rule is CASCADE.
 - v. DELETE part P2 in P, if the relevant delete rule is RESTRICT.
 - vi. DELETE project J4 in J, if the relevant delete rule is CASCADE.
 - vii. UPDATE shipment S1-P1-J1 in SPJ, setting SNO to S2. (shipment S1-P1-J1 means that in SPJ table the value for attributes SNO, PNO and JNO are S1, P1 and J1 respectively)
 - viii. UPDATE shipment S5-P5-J5 in SPJ, setting JNO to J7.
 - ix. UPDATE shipment S5-P5-J5 in SPJ, setting JNO to J8
 - x. INSERT shipment S5-P6-J7 in SPJ.
 - xi. INSERT shipment S4-P7-J6 in SPJ
 - xii. INSERT shipment S1-P2-jjj (where jjj stands for a default project number).
-
-
-

- 5) Find the name of projects in the relations above, to which supplier S1 has supplied using relational algebra.
-
.....
.....
.....

2.5 SUMMARY

This unit is an attempt to provide a detailed viewpoint of database design. The topics covered in this unit include the relational model including the representation of relations, operations such as set type operators and relational operators on relations. The E-R model explained in this unit covers the basic aspects of E-R modeling. E-R modeling is quite common to database design, therefore, you must attempt as many problems as possible from the further reading. The E-R diagram has also been extended. However, that is beyond the scope of this unit. You may refer to further readings for more details on E-R diagrams.

2.6 SOLUTIONS/ ANSWERS

Check Your Progress 1

1. Supplier Number (SNO), Supplier Name (SNAME) and City of the location of the supplier (CITY). The present instance s of the relation S has 2 tuples.
2. Domain of SNO is the codes that are being assigned to suppliers. The present coding uses the first character as S followed by the sequence number of a supplier. The domain of SNAME is strings of characters of some defined length and the domain of CITY is the set of possible cities in India.
3. The relation will not change; the order of tuples will change.
4. The super key of the relation could be (SNO, SNAME, CITY) or (SNO, SNAME) or (SNO, CITY) or assuming every supplier has unique name (SNAME, CITY) or SNO or assuming every supplier has unique name SNAME.

The possible candidate keys are SNO or assuming every supplier has unique name SNAME

Primary key may be selected as SNO and alternate key would be SNAME

Check Your Progress 2

1. Domain constraint for S will be for CITY which should be checked to be in a selected list of cities of India. SNO may be checked by a Range constraint and SNAME may be data type with maximum allowable length of name.
2. The Primary keys of relation S, P and J are SNO, PNO and JNO respectively. The primary key of SPJ is a composite key (SNO, PNO, JNO). As per the primary key constraint none of the attributes of primary key in SPJ, viz. SNO, PNO or JNO can be null.
3. There are three foreign keys, all of them are in SPJ relation. These are (i) SNO (ii) PNO (iii) JNO. SNO of SPJ references attribute SNO in S; PNO of SPJ references attribute PNO in P; and JNO of SPJ references attribute JNO in J. As per foreign key constraints the values of SNO, PNO and JNO in SPJ can be only as per the related referenced attributes.
4. The violations are as under:
 - a. Primary key of S is SNO, it cannot be Null, you may change it to <S4, Jack, Mumbai>, which will be inserted successfully in S.

- b. Primary key violation in SPJ as PNO cannot be Null as it is a part of primary key. There is another violation here, foreign key JNO has a value J3, which does not exist in JNO attribute of relation J for this instance, thus, the allowable values for JNO in SPJ is just J1 or J2. Thus, a correct record insertion in SPJ would be: <'S2', 'P3', 'J2', 200>.
- c. The primary key 'P2' already exists and cannot be duplicated, the correct insertion may be to Insert <'P4, 'Pencil', 'Grey', 'Kolkata'> into P.
- d. There is no violation.

Check Your Progress 3

1. relational algebra
2. $A - (A - B)$
3. (d) Join
4.
 - i. Accepted
 - ii. Rejected (candidate key uniqueness violation)
 - iii. Rejected (violates RESTRICTED update rule, as SPJ contains tuples having value S5 in SNO)
 - iv. Accepted (supplier S3 and all shipments for supplier S3 in the relation SPJ would be deleted, as the rule is CASCADE)
 - v. Rejected (violates RESTRICTED delete rule, as SPJ contains tuples having value P2 in PNO)
 - vi. Accepted (project J4 and all shipments for project J4 from the relation SPJ are deleted)
 - vii. Accepted
 - viii. Rejected (primary/candidate key uniqueness violation as tuple S5-P5-J7 already exists in relation SPJ)
 - ix. Rejected (referential integrity violation as there exists no tuple for J8 in relation J)
 - x. Accepted
 - xi. Rejected (referential integrity violation as there exists no tuple for P7 in relation P)
 - xii. Rejected (referential integrity violation – the default project number jjj does not exist in relation J).
- 5) The answer to this query will require the use of the relational algebraic operations. This can be found by performing selection of supplies made by S1 in SPJ, then taking projection of resultant on JNO and joining the resultant to J relation. Let us show steps:

First find out the supplies made by supplier S1 by selecting those tuples from SPJ where SNO is S1. The relation operator being:

$$SPJT = \sigma_{SNO = 'S1'} (SPJ)$$

The resulting temporary relation SPJT will be:

SNO	PNO	JNO	QUANTITY
S1	P1	J1	200
S1	P1	J4	700

Next, you may take projection of SPJT on PNO

$$SPJT2 = \Pi_{JNO} (SPJT)$$

The resulting temporary relation SPJT will be:

JNO
J1
J4

Next, take natural JOIN this table with J:

RESULT = SPJT2 \bowtie J

The resulting relation RESULT will be:

JNO	JNAME	CITY
J1	Sorter	Pune
J4	Console	Agra

You can write it as a single relational algebraic query as:

RESULT = ($\Pi_{JNO}(\sigma_{SNO = 'S1'} (SPJ))$ \bowtie J)



UNIT 3 ENTITY RELATIONSHIP MODEL

- 3.0 Introduction
- 3.1 Objective
- 3.2 Entity Relationship (E-R) Model
 - 3.2.1 Entities
 - 3.2.2 Attributes
 - 3.2.3 Relationships
 - 3.2.4 E-R diagram Basics
 - 3.2.5 More about Relationships
 - 3.2.6 Extended E-R Features
 - 3.2.7 Defining Relationship for College Database
- 3.3 An Example
- 3.4 Conversion of E-R Diagram to Relational Database
- 3.5 Enhanced E-R Model
- 3.6 Converting E-R and EER Diagram to Relations
- 3.7 Summary
- 3.8 Solution/Answers

3.0 INTRODUCTION

In the previous unit of this block, you have gone through the concept of relational database management systems and one of the important languages for relational database – relational algebra. This unit explains, you about of an analysis model of the database system, known as Entity-Relationship (E-R) model. The E-R model is a widely used model for analysis of data requirements of an organisation. The E-R model is primarily a semantic model and is very useful in creating raw database design that can be further normalised. With the availability of object-oriented technologies, the E-R model has been extended to include object-oriented features. This unit also discusses these E-R extensions. We will also discuss the conversion of E-R diagrams to tables, in this unit.

3.1 OBJECTIVES

After going through this unit, you should be able to:

- Define and explain various components of E-R model;
- draw an E-R diagram for a given problem;
- convert an E-R diagram to a relational database;
- Explain the role of extended features of E-R model;
- Convert an EER diagram to relations.

3.2 ENTITY RELATIONSHIP (E-R) MODEL

Some of the important characteristics of E-R model are listed below:

- *Entity relationship model is a high-level conceptual data model.*
- *It allows you to describe the data involved in a real-world enterprise in terms of entities and their relationships.*
- *It is widely used to create an initial design of a database.*
- *It provides a set of useful concepts that make it convenient for a developer to move from a basic set of information to a detailed and precise description of information that can be easily implemented in a database system.*
- *It describes data as a collection of entities, relationships and attributes.*

In the following sections, we explain the basic terms used in the E-R model.

3.2.1 Entities

First let us answer the question: **What are entities?**

- An entity is an object of concern, which is used to represent the things in the real world, e.g., car, table, book, etc.
- An entity need not be a physical entity, it can also represent a concept in the real world, e.g., project, loan, etc.
- It represents a class of things, not any one instance, e.g., ‘STUDENT’ entity has instances of ‘Ramesh’ and ‘Mohan’.

Entity Set or Entity Type: A collection of a similar kind of entity is called an Entity Set or entity type.

For the COLLEGE database, the objects of concern are Student, Faculty, Course and Department. The collections of all the students’ entities form an entity set STUDENT. Similarly, collection of all the courses form an entity set COURSE.

You may please note that entity sets need not be disjoint. For example – an entity, say Mohan, may be part of the entity set STUDENT, the entity set FACULTY and the entity set PERSON.

Entity identifier - key attributes: An entity set usually has one or more attributes, which attains unique value for every distinct entity in a given entity set. Such an attribute or set of attributes is/are called key attribute(s) and its values can be used to identify each entity uniquely in the given entity set.

Strong entity set: An entity set which contains at least one key attribute is a Strong entity set. For example, a Student entity set would contain at least one key attribute *Enrolment number*, which is unique for every student, thus, the entity set Student is a strong entity set.

Weak entity set: Entity sets that do not contain any key attribute, and hence cannot be identified independently, are called weak entity sets. A weak entity cannot be identified uniquely by its attributes, therefore, are recognised in conjunction with the

primary key attributes of another strong entity on which its existence is dependent (called owner entity set).

Generally, a primary key of an owner entity set is attached to a weak entity set, which has identifying attributes, called discriminator attributes. These two together form the primary key of the weak entity set. The following restrictions must hold for the above:

- The owner entity set and the weak entity set must participate in one to many relationship set. This relationship set is called the identifying relationship set of the weak entity set.
- The weak entity set must have total participation in the identifying relationship.

One of the most common examples about the weak entity set is an entity set Dependent and the related Strong entity set Employee in an organisation. The Dependent entity set is used to list all the dependents of each employee of an organisation. The attributes of the Dependent entity set are: Dependent name, birth date, gender and relationship with the employee. Each Employee entity is said to own the dependent entities that are related to it. However, please note that the 'Dependent' entity does not exist of its own, it is dependent on the Employee entity. In other words, you can say that in case an employee leaves the organization, all dependents related to him/her also leave along with this employee. Thus, a 'Dependent' entity has no significance without the entity 'Employee'. Thus, it is a weak entity.

3.2.2 Attributes

Let us first define - **What is an attribute?**

An attribute is an element of an entity, which can contain a representative value. In other words, an entity is represented by a set of attributes.

For example, a Student entity set may consist of attributes - Roll no, student's name, age, address, course, etc. An entity will have a value for each of its attributes. For example, for a particular student, the following values can be assigned:

Roll No:	1234
Name:	Mohan
Age:	18
Address:	Z-894, Maidan Garhi, Delhi.
Course:	B.Sc. (H)

Domains: Each simple attribute of an entity type contains a possible set of values that can be attached to it. This is called the domain of an attribute. An attribute cannot contain a value outside this domain.

EXAMPLE- for STUDENT entity Age has a specific domain, integer values say from 15 to 90.

Types of attributes

Attributes attached to an entity can be of various types. They are explained below:

Simple: An attribute that cannot be further divided into smaller parts and represents the basic meaning is called a simple attribute. For example: Each of the attributes - 'FirstName', 'LastName', age of PERSON entity set are simple attributes.

Composite: Attributes that can be further divided into smaller units and each smaller unit contains specific meaning. For example, the attribute NAME of a FACULTY entity can be subdivided into First name, Last name and Middle name.

Single valued: Attributes having a single value for a particular entity. For Example, Age is a single valued attribute of a STUDENT entity.

Multivalued: Attributes that have more than one value for a particular entity is called a multivalued attribute. Different entities may have different numbers of values for these kinds of attributes. For multivalued attributes you must also specify the minimum and maximum number of values that can be attached. For example, the phone number for a PERSON entity is a multivalued attribute.

Stored and derived: Attributes that are directly stored in the database are called stored attributes. For example, 'Birth Date' attribute of a PERSON entity can be a stored attribute. However, there are certain attributes, whose value can be computed from the value of the stored attribute. For example, in the PERSON entity, the attribute 'Birth Date' can be used to compute the attribute Age of a person on a specific day. Thus, 'Birth Date' is a stored attribute, whereas Age may be a derived attribute for this entity.

3.2.3 Relationships

First, let us define the term relationships, i.e. **What Are Relationships?**

A relationship can be defined as:

- a connection or set of associations, or
- a rule for communication among entities:

Example: In a COLLEGE database, the association between student and course entity set, i.e., in the statement "Student **opts** Course" **opts** is an example of a relationship between the two entities Student and Course.

Relationship Sets

A relationship set is a set of relationships of the same type. For example, consider the relationship between two entity sets STUDENT and COURSE. Collection of all the instances of relationship **opts** forms a relationship set.

3.2.4 E-R Diagram Basics

The logical structure of a database is modeled using an E-R model, which is graphically represented with the help of an E-R diagram. The basic symbols used in an E-R diagrams are given in the following table:

Object	Represented by
Entity Set	Rectangle
Attribute	Ellipse
Relationship Set	Diamonds

Figure 3.1 shows different kinds of entities, attributes, relationships and participation constraints, which are explained in this Unit.

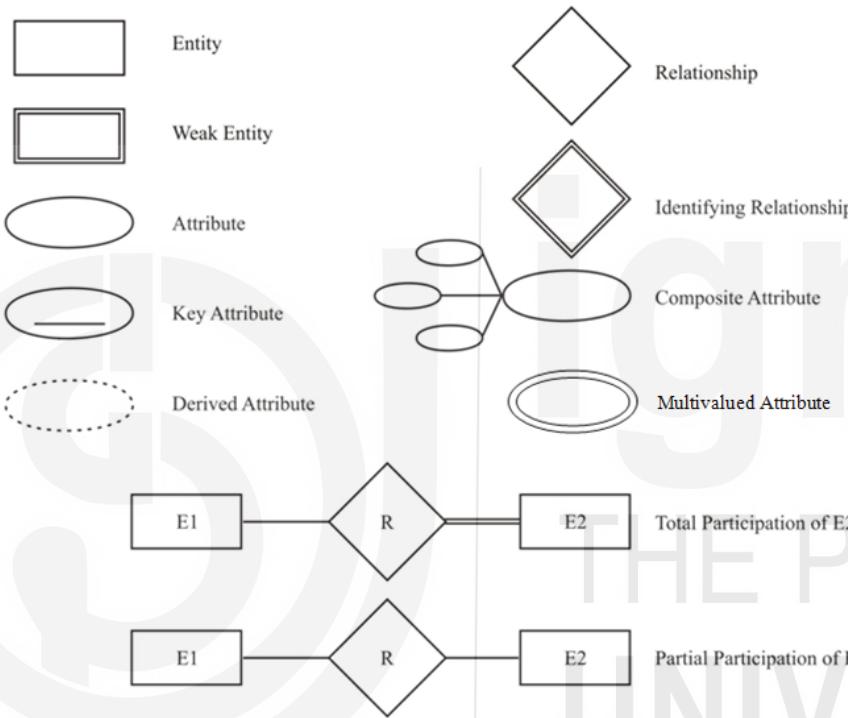


Figure 3.1: Symbols of E-R diagrams

3.2.5 More about Relationships

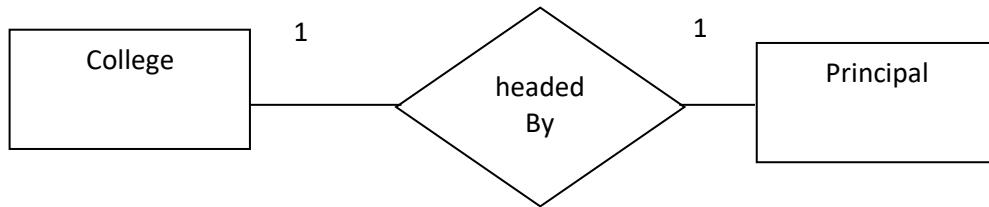
In this section, you will go through some of the important concepts, which are used for making good E-R models.

Degree of a relationship set: The degree of a relationship set is the number of participating Entity sets. The relationship between two entities is called a binary relationship. A relationship among three entities is called a ternary relationship. Similarly, a relationship among n entities is called an n-ary relationship.

Cardinality of a relationship set: Cardinality specifies the number of instances of an entity associated with another entity participating in a relationship. Based on the cardinality, binary relationships can be further classified into the following categories:

- **One-to-one:** An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A.

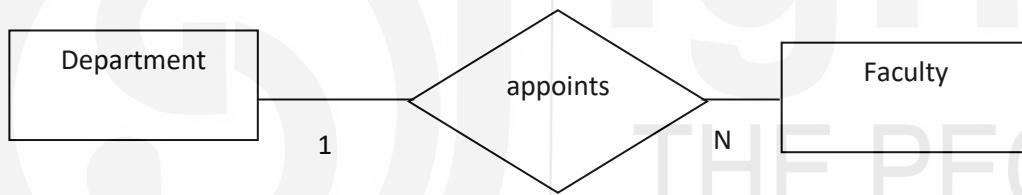
For example, the relationship *headedBy* between college entity set and principal entity set would be one-to-one, as one college can have at most one principal; and one principal can be principal of only one college. (This example assumes that a principal can be head of only one college.)



Similarly, you can define the relationship between University and Vice-Chancellor.

- *One-to-many*: Consider entity sets A and entity set B has a relationship cardinality A : B, as 1:N. This suggests that one specific entity of A may be related with several entities of entity set B.

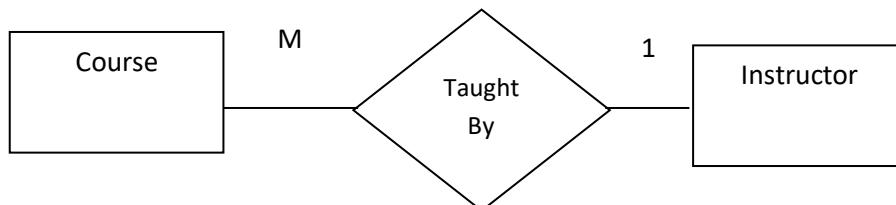
For example, relationship between Department entity set and Faculty entity set (assuming that a faculty member can work in only one department).



For example, in the diagram above, several faculty members may be appointed in one department, however, a specific faculty member will be appointed in precisely one department.

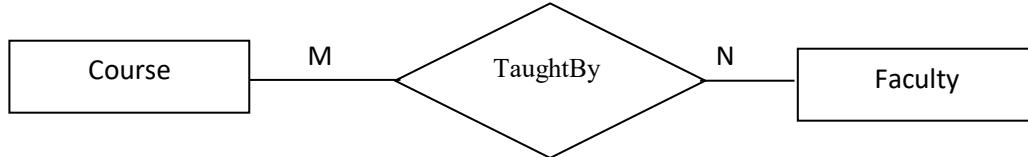
- *Many-to-one*: Consider entity sets A and entity set B has a relationship cardinality A : B, as N : 1. This suggests that several entities of A may be related with one specific entity of entity set B.

For example, the relationship between entity set Course and entity set Instructor. An instructor can teach various courses, but a single course can be taught only by one instructor. Please note this is an assumption.



Many-to-many: Entities in entity set A and entity set B are associated with any number of entities from each other.

For example, consider that a course can be taught jointly by many faculty members and each faculty member can teach several courses, then many-to-many relationship holds, as shown below:



Another example is shown in the diagram given below. The relationship cardinality M : N. This implies that an Author entity can be correlated to many Book entities, which are written by him/her. Further, a Book entity can also be correlated with several Author entities who have written the Book.



Recursive relationships: A recursive relationship is that relationships in which both the participating entity sets are the same entity set, however, the role of the entity set in each participation is different.

Participation constraints: The participation Constraints specify whether the existence of an entity depends on its being related to another entity via the relationship type. There are two types of participation constraints:

Total: When all the entities from an entity set participate in a relationship set, it is termed as the *total* participation. For example, the participation of the entity set Course in the relationship set ‘TaughtBy’ can be said to be *total* because every Course must be taught by a faculty.

Partial: When it is not necessary for all the entities from an entity set to participate in a relationship set, it is termed as *partial* participation. For example, the participation of the entity set Instructor in the relationship set ‘TaughtBy’ can be partial, since not every Instructor may be teaching a course.

3.2.6 Extended E-R Features

Although, the basic features of E-R diagrams are sufficient to design many database situations. However, with more complex relations and advanced database applications, it is required to use extended features of E-R models. The three such features are:

- Generalisation
- Specialisation, and
- Aggregation

We have explained them with the help of an example. More details on them are available in the further readings.

Example 1: A bank has an Account entity set. Any accounts of the bank can be one of two types: (1) Savings account and (2) Current account.

The statement above represents a specialisation/ generalisation hierarchy. It can be shown as:

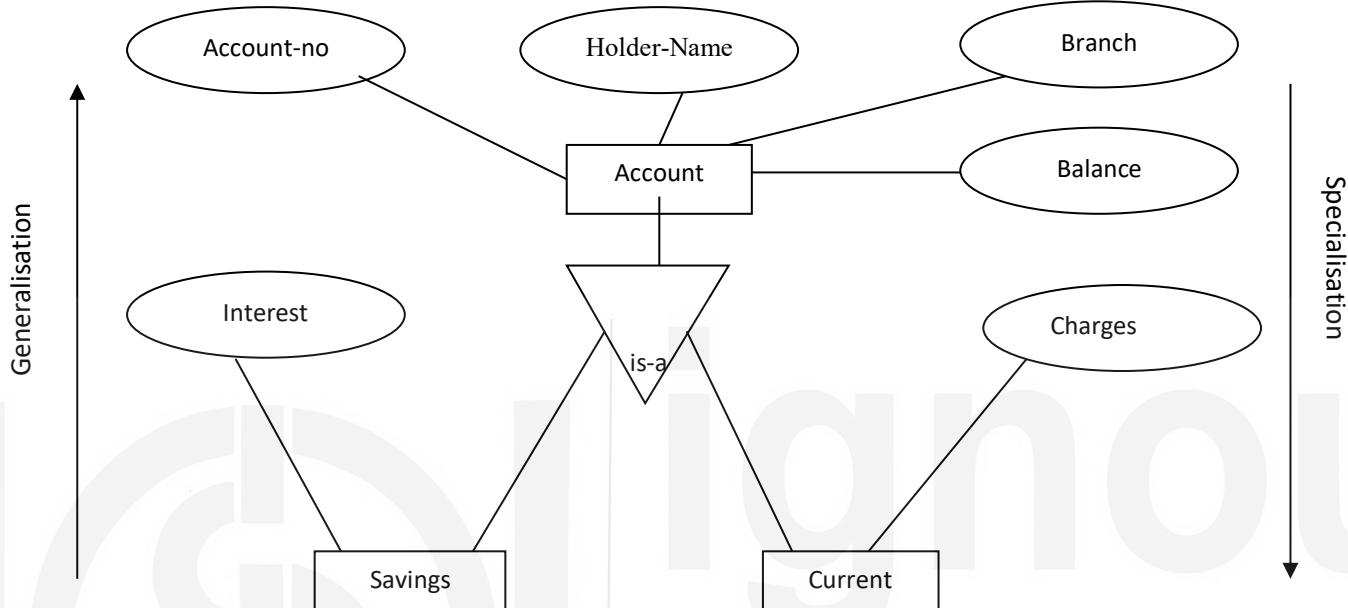


Figure 3.2: Generalisation and Specialisation hierarchy

Aggregation: One limitation of the E-R diagram is that they do not allow representation of relationships among relationships. In such a case the relationship along with its entities are promoted (aggregated to form an aggregate entity which can be used for expressing the required relationships). A detailed discussion on aggregation is beyond the scope of this unit you can refer to the further readings for more detail.

3.3 AN EXAMPLE

Let us explain it with the help of an example application. We will describe here an example database application of a COLLEGE database and use it for illustrating various E-R modeling concepts.

Problem Statement

A college database keeps track of Students, faculty, Departments and Courses. Following paragraphs gives the description of a COLLEGE database system.

A College contains various departments like Department of English, Department of Hindi, Department of Computer Science etc. Each department is assigned a unique

id and name. Some faculty members are appointed to each department and one of them works as head of the department.

There are various courses conducted by each department. Each course is assigned a unique id, name and duration.

The information contained for various objects are stated below:

- Faculty information contains name, address, department, basic salary etc. A faculty member is assigned to only one department but can teach courses of another department.
- Student's information contains Roll no (unique), Name, Address etc. A student can opt only for one course and one department only.
- Student's Parent or Guardian information to be recorded is - name of parent or guardian, age of the parent or guardian, gender and address of parent or guardian.

A student is allowed to take only one course. A student is assisted by his/her guardian. A faculty works in a department, which is headed by a faculty member. A course is taught by a faculty, who is allowed to teach several courses.

Defining Entities and Attributes

One of the simplest ways to identify the entities is to look for the proper nouns. This gives us possible set of entities in the problem statement are:

Student, Faculty, Department, Course, Guardian

The attributes of these entities can be found from the statements. You may also note except of Guardian:

Student (Rollno – Primary Key, Name, Address)
Faculty (Id – Primary Key, Name, Address, Basic_Sal)
Department (D_No, - Primary Key, D_Name)
Course (Course_ID – primary key, Course_Name, Duration)
Guardian (Name, Address, Relationship)

Please note that the Guardian is a weak entity. It is related to the strong entity Student.

Defining Relationship

Using the concepts defined earlier, we have identified that strong entities in COLLEGE database are Student, Faculty, Course and Department. This database also has one weak entity called Guardian. You can specify the following relationships among these entities. Further, these relationships also show the relationship cardinality and participation constraints:

1. **Head_of** is a 1:1 relationship between Faculty and Department. Participation of the entity Faculty is *partial* since not all the faculty members participate in this relationship (as all cannot be the head), while the participation from the Department side is *total* since every department has one head. Please also note that this relationship has an attribute Date_from, which stores the date from which the given appointment was applicable.

2. **Works_in** is a 1:N relationship between Department And Faculty, as one department can have many faculty, whereas a faculty is associated with only one department. Participation from both the sides is total, as every department has at least one faculty and every faculty must belong to a department.
3. **Opts** is a 1:N relationship between Course and Student. Participation from the Student side is *total* because we are assuming that each student opts for one course. But the participation from the course side is *partial*, since there can be courses that have no students.
4. **Taught_by** is a M: N relationship between Faculty and Course, as a Faculty can teach many courses and a Course can be taught by many faculty members.
5. **Enrolled** is a 1:N relationship between Student and Department as a student is allowed to enroll for only one department at a time. A student must enroll in a Department. However, a newly created Department may not have a student.
6. **Assisted_by** is a 1:N relationship between Student and Guardian as a student can have more than one local guardian and one local guardian is assumed to be related to one student only. The weak entity Guardian has *total* participation in the relation “Assisted_by”.

Now, you are ready to make an E-R diagram for the college database. The E-R diagram.

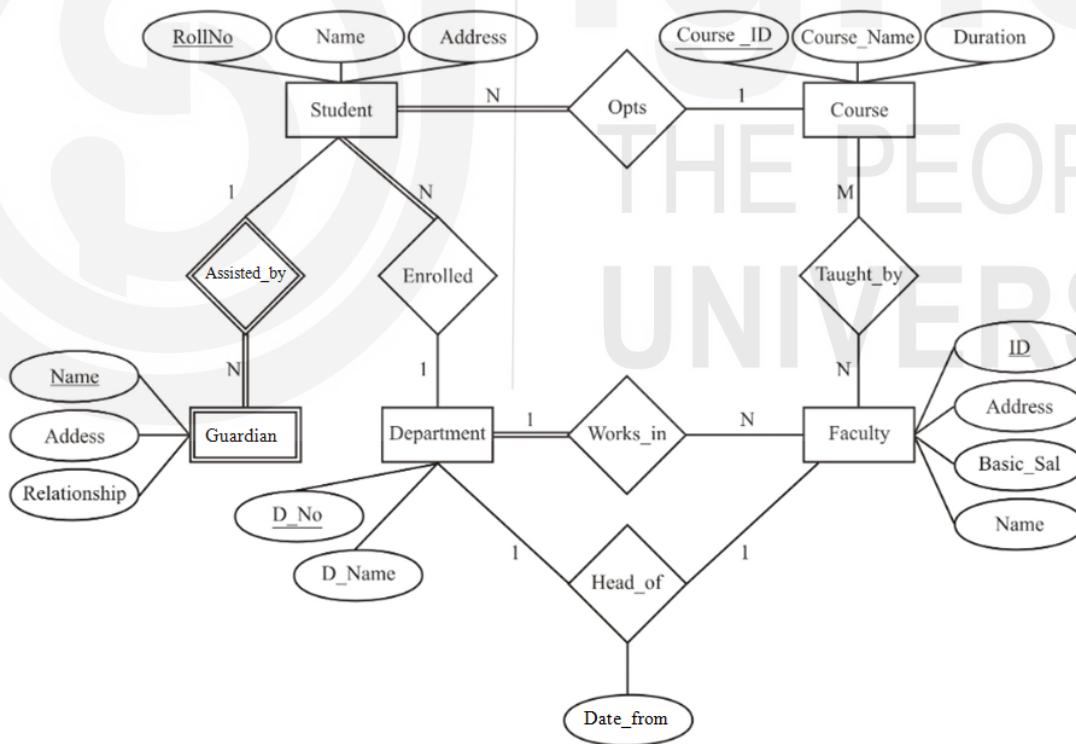


Figure 3.3: E-R diagram of COLLEGE database

3.4 CONVERSION OF E-R DIAGRAM TO RELATIONAL DATABASE

A relational database management system is designed from an E-R diagram, which represents various entities and relationships among entities for an application using the following methods.

Conversion of entity sets:

Strong entity set: For each strong entity set E in the E-R diagram, you create a relation R containing all the simple attributes of E. The primary key of the relation R will be one of the key attributes of R.

For example, the entities Student, Faculty, Course and Department, which are strong entities, relations as shown in Figure 3.4 would be created.

Student (Rollno, Name, Address)
Faculty (Id, Name, Address, Basic_Sal)
Department (D_No, D_Name)
Course (Course_ID, Course_Name, Duration)

Figure 3.4: Conversion of Strong Entities to relations

II) For each weak entity type W in the E-R Diagram, you create another relation R that contains all simple attributes of W. Further, you add the key attribute(s) of the owner entity set (say KP) of W in R. The primary key to this relation R is – <KP + Discriminator attribute of W> and foreign key is KP, which references the owner entity of W.

For example, conversion of weak entity Guardian into relation is shown in Figure 3.5. Please note that the owner entity of the Guardian entity is the strong entity Student, whose key is RollNo. Therefore, the key to Guardian relation is RollNo+Name. The Foreign key in Guardian relation is RollNo, which refers to Student relation.

Guardian (RollNO, Name, Address, Relationship)
Foreign Key: RollNO refers to relation Student

Figure 3.5: Conversion of weak entity Guardian to relation.

Conversion of relationship sets

Binary Relationships

I) One-to-one relationships:

For each 1:1 relationship set in the E-R diagram involving two entities E1 and E2 you choose one of the two entities (say E1), preferably the one with total participation, and add the primary key attribute of the other entity E2, in the relation of entity E1. Make this added attribute in E1 relation as a foreign key attribute to the relation

created from another entity (E2). You should also include all the simple attributes of the relationship type, if any, in the relation E1.

For example, the Head_of relationship in Figure 3.3 is 1:1. The two entities participating in the relationship are - Department and Faculty. The participation of the Department entity is *total* in the Head_of relationship. Therefore, the ID attribute, which is the primary key of the Faculty entity is added to Department relation. Please note, you can rename this attribute in the Department relation, if needed. In addition, this attribute in Department relation will be the foreign key to the Faculty relation. Further, the attribute Date_from of the Head_of Relationship is also added to the Department relation. This is shown in Figure 3.6. Please note that you will keep information in this the relation only stores the ID of the current head and Date from which s/he is the head. Please also note that you will not create a separate table of the relationship Head_of.

Department (D_No, D_Name, Head_ID, Date_from))

The ID attribute of Faculty relations is added to Department relation and has been renamed as Head_ID.

Foreign Key: Head_ID references ID attribute in Faculty relation

Figure 3.6: Converting 1:1 relationship (No new relation is added)

II) One-to-many or many-to-one relationships:

Both relationship sets involve two entity sets, say E1 and E2. Further, assume that E1 is on the many side and E2 is on the one side of the relationship set. You just need to include the primary key of the relation of entity on the one side (E2 in the case as above) to the relation created for the entity set of many side (E1 in the present case). You should include all simple attributes (or simple components of a composite attributes of relationship set, if any) in the relation of E1. Please note that now the relation of E1 has a foreign key reference to the relation of E2.

For example, the Works_in relationship between the entities Department and Faculty. For this relationship, the entity at N side is Faculty, therefore add primary key attribute of entity Department, i.e., D_No as a foreign key attribute in relation created for Faculty entity set. This is shown in Figure 3.7

Faculty (Id, Name, Address, Basic_Sal, D_No))

The Works_in relationship has been included in Faculty relation.

D_No is a foreign key and references the relation Department.

Figure 3.7: Converting 1:N relationship to relation (No new table is added in this case)

III) Many-to-many (M : N) relationship:

Consider that a M : N relationship set is binary with two participating entities, say Entity1 and Entity2. For this type of relationship set a relation is created. This relationship set should contain the Primary key of Entity1 (say PKE1), as well as the Primary key of Entity2 (say PKE2). In addition, any attribute of the relationship set is added to the relation. The primary key of this newly formed relation of the M : N relationship set is the composite primary key PKE1+PKE2. Please also note that for this new relation of the relationship set, there exists two foreign keys – PKE1, which refers to the relation of Entity1 and PKE2, which refers to the relation of Entity set Entity2.

For example, the m : n relationship Taught_by between entity sets Course and Faculty should be represented as a new table. The structure of the table will include the primary key of Course and primary key of Faculty entities. Please note that both Course and Faculty relations remain unchanged.

Add the following relation into already existing list of relations:

Taught_by (Course_ID, ID)

Primary Key: Course_ID + ID

Foreign Keys:

Course_ID references Course relation

ID references Faculty relation

This relation has no other attribute, as the relationship has no attribute.

Figure 3.8: Converting m : n relationship Taught_by into relations

N-ary Relationships

There are several cases for creating relations for n-ary relationships. A very general case is presented here. For each n-ary relationship set R where $n > 2$, you create a new table S to represent R. You should include the primary key of all the participating entity sets as the foreign key attributes in S. You should include any simple attributes of the n-ary relationship set (or simple components of complete attributes) as attributes of S. The primary key of S is usually a combination of all the foreign keys that reference the relations representing the participating entity sets. Figure 3.8 is a special case of n-ary relationship, i.e. a binary relationship.

Multivalued attributes:

For each multivalued attribute ‘A’ of an entity set E, you can create a new relation R that includes an attribute corresponding to the primary key attribute of the relation entity E that represents the entity set or relationship set that has as an attribute. The primary key of R is then a combination of A and the primary key of relation of E. For example, if a Student entity had RollNo, Name and PhoneNumber attributes, where phone number is a multivalued attribute, then you will create two tables for this entity as given below:

Student (RollNo, Name)

Phone (RollNo, PhoneNumber)

Converting Generalisation / Specialisation hierarchy to tables:

A simple rule for conversion may be to decompose all the specialised entities into relations in case they are disjoint. For example, for the E-R diagram of Figure 3.1, you can create the two tables as:

Saving_account (account-no, holder-name, branch, balance, interest).

Current_account (account-no, holder-name, branch, balance, charges).

The other way might be to create tables that are overlapping (not disjoint) for example, assuming that in the E-R diagram of Figure 3.2 contains overlapping sub-classes, then you would be creating the following three relations:

The first relation would be for the higher level entity:

account (account-no, holder-name, branch, balance)

The specialisation entities will contain the Primary key of the generalised entity and all the attributes of entity itself, as shown below:

saving (account-no, interest)

current (account-no, charges)

Thus, the information about a single account can be found in all the three relations.

Check Your Progress 1

- 1) A company wants to develop an application to store information about its clients. Find the possible entities and relationships among the entities. Show the step-by-step procedure to make an E-R diagram for the application.

.....
.....
.....
.....

- 2) An employee works for a department. If the employee is a manager, then s/he manages the department. Every employee works on at least one or more projects, which are controlled by various departments of a company. An employee can have many dependents. Draw an E-R diagram for the above company. Find all possible entities and their relationships.

.....
.....
.....
.....

- 3) A supplier, located in only one-city, supplies various parts for the projects of different companies located in various cities. You can name this database as “supplier-and-parts”. Draw the E-R diagram for the supplier-and-parts.

.....
.....
.....

- 4) Convert the E-R diagram created for question 2 above into a relational database.

.....
.....

3.5 ENHANCED E-R MODEL

Enhanced E-R models can help in designing of relational and object-relational database systems. In addition, to E-R modeling concepts, the Enhanced ER model includes:

- 1) Subclass and Super class
- 2) Inheritance
- 3) Specialisation and Generalisation.

As discussed earlier, an entity may be an object with physical existence or it may be an object with a conceptual existence. An entity is depicted by a set of attributes. Sometimes, an entity can consist of explicit sub-groupings. For example, an entity *vehicle* consists of sub-groups – Truck (or Commercial Vehicles), Light Motor Vehicles (or Car), Two-Wheelers (or Scooter), etc. Every sub-grouping must belong to entity set *vehicle*, therefore, these sub-groupings are called a subclass of the *vehicle* entity set and the *vehicle* itself is called the super class for each of these subclasses.

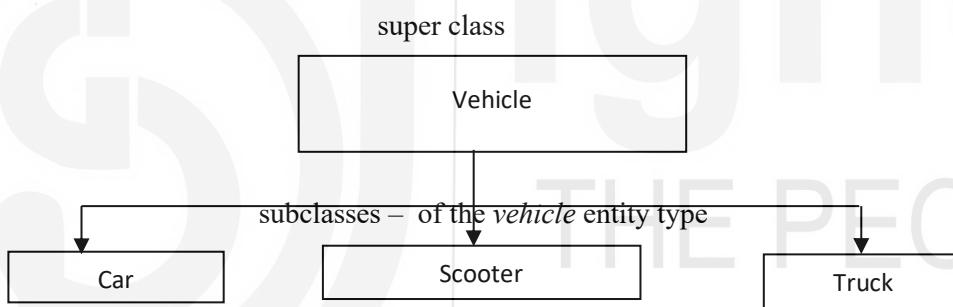


Figure 3.9: A class hierarchy

The relationship between a super class and any of its subclasses is called class/subclass relationship. It is often called an IS-A relationship because of the way we refer to the concept, as you would write, “Car *is-a* vehicle”. The member entity of the sub-class represents the same real world as the member entity of the super class. If an entity is a member of a sub-class, by default it must also become a member of the super class; whereas it is not necessary that every entity of the super class must be a member of its sub-class. An entity that is a member of a sub-class inherits all the attributes of its super class. An entity set is identified by its attributes and the relationship sets in which it participates; therefore, a sub-class entity also inherits all the relationships in which the super class participates. According to **inheritance** the sub-class inherits attributes and relationships of the super class. In addition, every subclass can contain its own attributes and relationships in which it has participated.

Specialisation is a process in which an entity set (super class) is modeled as a set of sub-entity sets (sub-classes) by using a specific distinguishing characteristic of the

super class. For example, in Figure 3.9, the super class *vehicle* is modeled using sub classes - Truck (or Commercial vehicle), Car (or Light Motor Vehicle), Scooter (or Two-wheelers)) using the *Type* attribute of the *vehicle* class. Please note that several specialisations hierarchies can be modeled using a super class by using different distinguishing characteristics. Figure 3.10 shows how you can represent a specialisation with the help of an EER diagram.

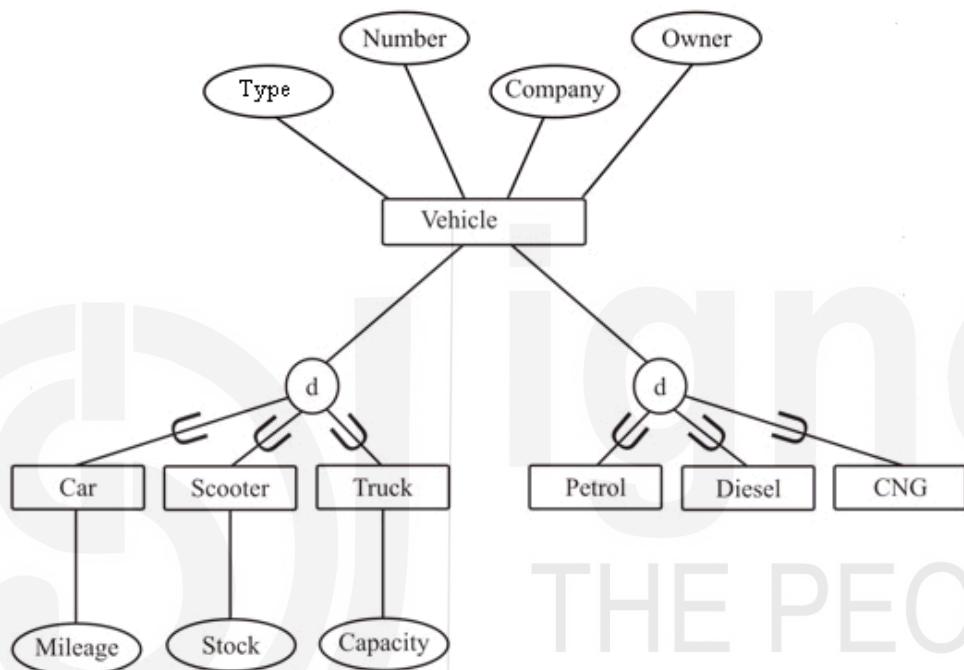


Figure 3.10: EER diagram showing more than one specialization from one super class

In Figure 3.10, letter 'd' in the circle indicates that all these subclasses are **disjoint** in nature, i.e. all the vehicle entities are disjoint, as they can be part of only one of the subclass. Please also notice that in Figure 3.10, common attributes, like vehicle number, owner name etc., are attributes of the super class, whereas attributes like mileage of car, stock of scooter and capacity of truck are the attributes of the sub-classes. Please note that an entity will be appearing twice in the EER diagram – once in the sub-class and the other in the super class (Please refer to Figure 3.11).

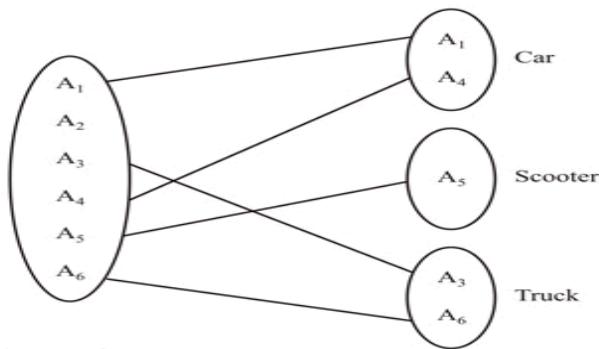


Figure 3.11: Sharing of members of the super class vehicle and its sub-classes

Hence, the specialisation is a set of sub-classes of an entity set, which establishes additional specific attributes with each sub-class and establishes additional specific relationship sets between a sub-class and other entity types or other sub-classes.

Generalisation is the reverse process of specialisation; in other words, it is a process of representing entity sets consisting of entities, which have almost similar attributes excepting few. The similar attributes of these entity set form the super class. For example, the entity set Car and Truck can be generalised into entity set Vehicle. Therefore, Car and Truck can now be sub-classes of the super class generalised class Vehicle (Refer to Figure 3.12).

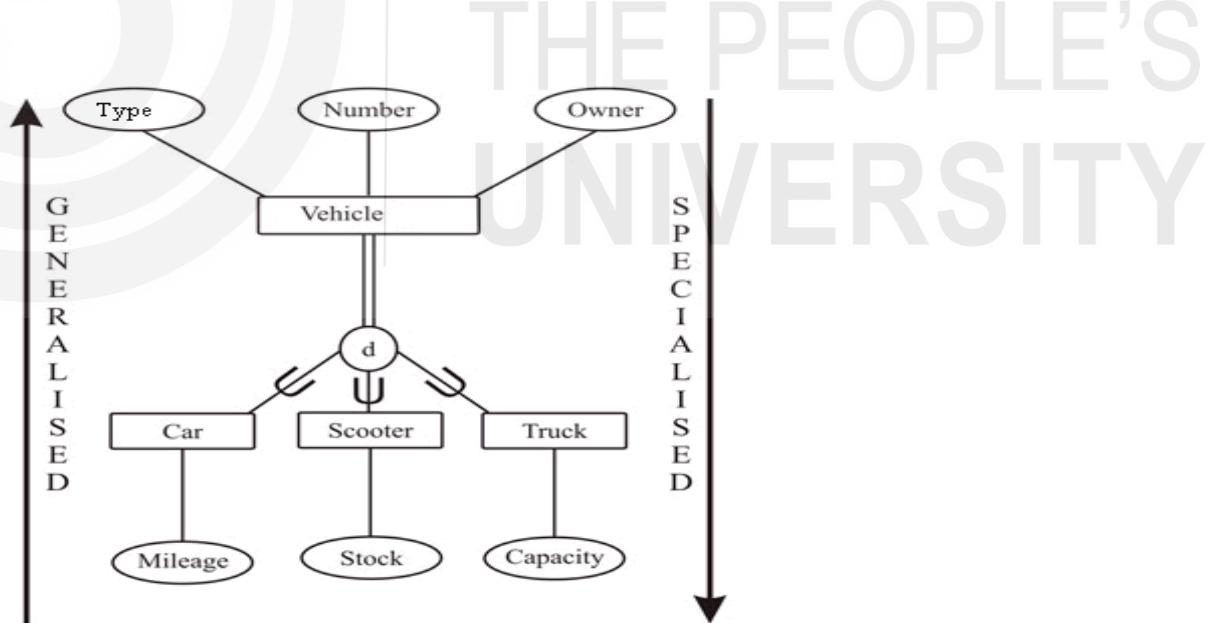


Figure 3.12: Generalisation and Specialisation

Constraints and Characteristics of Specialisation and Generalisation: A super class may either have a single sub-class or many sub-classes in specialisation. In

case of only one sub-class you do not use circle notation to show the relationship of sub-class/super class. Sometimes in specialisation, the subclass becomes the member of the super class after satisfying a condition on the value of some attributes of the super class. Such sub-classes are called *condition-defined* sub-classes or predicate defined subclasses. For example, the Vehicle entity set has an attribute vehicle “Type”, as shown in Figure 3.12.

You can specify the condition of membership for a subclass – car, truck, scooter – by the predicate – vehicle ‘Type’ of the super class vehicle. Therefore, a vehicle object can be a member of the sub-class, if it satisfies the membership condition for that sub-class. For example, to be a member of sub-class Car a vehicle entity must have the condition vehicle “type = car” as true. A specialisation in which an attribute or a set of attributes of the super class (called the *defining attribute* of specialisation) specify membership condition of the sub-classes, is termed as *attribute-defined* specialisation. In case no membership condition is stated for specialisation, a database user determines the membership. Such specialisation is termed as user-defined specialisation.

Disjointness is also the constraints to a specialisation, which specifies that a given entity cannot be a member of more than one sub-classes for a specific specialisation hierarchy. For example, in Figure 3.12, the symbol ‘d’ in circle stands for disjointness, as an entity can be a member of a single sub-class only. But if the real-world entity is not a disjoint entity sets of the sub-classes may overlap. This is represented by an (o) in the circle. For example, if you classify a class Book into sub-classes Textbooks and Reference Books, then you may like to define a specific book in both the sub-classes. This is a case of Overlapping constraints.

When every entity in the super class must be a member of some subclass in the specialisation it is called total specialisation. But if every entity does not necessarily need to belong to any of the subclasses, it is called partial specialisation. The total is represented by a double line. This is to note that in specialisation and generalisation the deletion of an entity from a super class implies automatic deletion from sub-classes belonging to the same; similarly, insertion of an entity in a super class results in insertion of the entity in all the sub-classes for which the attributes of this entity fulfills the constraints of attribute-defined specialisation. In case of total specialisation, insertion of an entity in a super class implies compulsory insertion in at least one of the sub-classes of the specialisation.

Union: In some cases, a single class has a similar relationship with more than one class. For example, the sub class ‘Car’ may be owned by two different types of owners: Individual or Organisation. Both these types of owners are different classes; thus, such a situation can be modeled with the help of a Union (Refer to Figure 3.13).

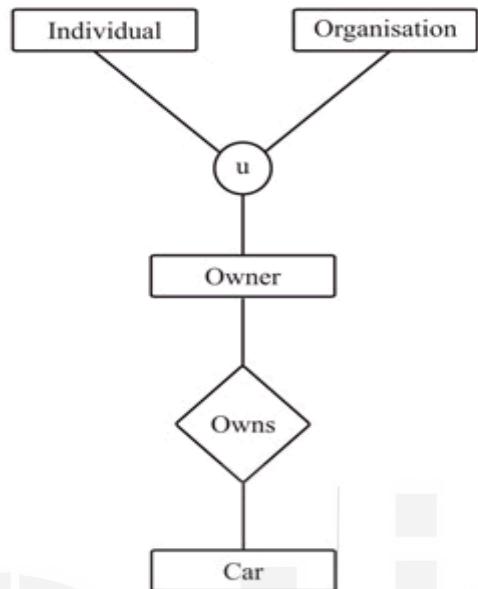


Figure 3.13: Union of classes

In the next section, we discuss how these extended features can be converted to relations.

3.6 CONVERTING EER DIAGRAM TO RELATIONS

The rules for converting the EER diagram, which primarily includes specialisation and generalisation hierarchy are the same as in the E-R diagram. Let us recapitulate these rules:

- Create a relation for each strong entity set.
- Create a relation for a weak entity set. The primary key of this relation would be a composite key involving the attributes of the primary key of the strong entity set on which it depends and discriminator of the weak entity.
- Create a relation for each binary $m : n$ relationship set having the primary keys of both the participating entities, which form the composite primary key to the relation. The individual primary keys are the foreign keys to respective relations.
- For a binary $m : 1$ or $1 : m$ relationship, in general, the primary key on the m side is added to 1 side of the entity. This also becomes the foreign key. For a binary $1:1$ relationship set the primary key of chosen participating relation is added to the other participating relation.
- Composite attributes may sometimes be converted to a separate relation.

- For generalisation or specialisation hierarchy a relation can be created for higher level and each of the lower-level entities. The higher-level entity would have the common attributes and each lower-level relation would have the primary key of the higher-level entity and the attributes defined at the lower specialised level. However, for a complete disjoint hierarchy no relation may be made at the higher level, but the relations are made at the lower level including the attributes of higher level.
- For an aggregation, all the entities and relationships of the aggregation are transformed into the relation on the basis of rules stated above. A relation is also created for the relationship set that exists between the aggregated entity (say AgE) and another simple entity (SE). The primary key of this new relation is the composite key involving the primary key of the AgE and SE.

So let us now discuss the process of converting the EER diagram into a relation. In case of disjoint constraints with total participation. It is advisable to create separate relations for the sub-classes. But the only problem in such a case will be to implement the referential entity constraints suitably.

For example, assuming that this EER diagram can be converted into a relation as:

Car (Number, owner, mileage)
 Scooter (Number, owner, stock)
 Truck (Number, owner, capacity)

Please note that referential integrity constraint in this case would require a relationship with three relations and therefore is more complex to implement.

In case, in the EER diagram of Figure 3.12 there is NO total participation of Vehicle super class in the sub-classes, then there will be some vehicles, which are not Car, Scooter and Truck, so how can you represent these? In addition, in case of overlapping constraints, some tuples may get represented in more than one relation. Thus, in such cases, it is ideal to create one relation for the super class and other relations for the sub-classes having the primary key and any other attributes of that sub-class. For example, with NO total participation the following relations would be created for the EER diagram of Figure 3.12:

Vehicle (Number, owner, type)
 Car (Number, mileage)
 Scooter (Number, stock)
 Truck (Number, capacity)

Finally, in the case of union since it represents dissimilar classes, you may represent separate relations. For example, both individual and organisation will be modeled to separate relations.

Check Your Progress 2

- 1) What is the use of the EER diagram?

.....

.....

.....

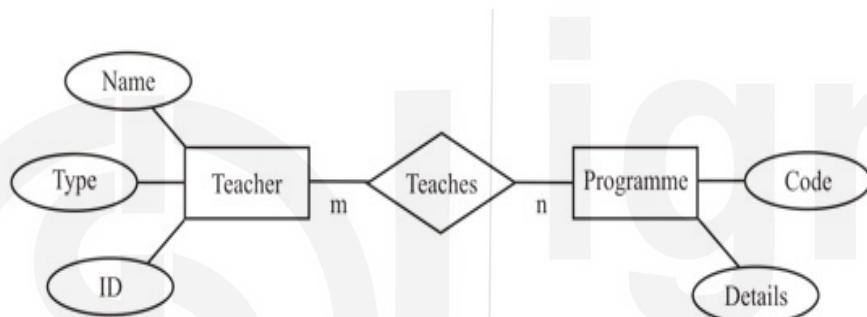
- 2) What are the constraints used in EER diagrams?

.....
.....
.....

- 3) How is an EER diagram converted into a relation?

.....
.....
.....

- 4) Consider the following E-R diagram.



'Type' can be regular or visiting faculty. Visiting faculty members can teach only one programme. Make a suitable EER diagram for this and convert the EER diagram to table.

.....
.....
.....

3.7 SUMMARY

This unit presents the concept of E-R model and EER models. Both these models are represented with the help of E-R diagram and EER diagram. These diagrams are very powerful tools to represent the need of data in a database system and can be used for the design of a good database system. The E-R model explained in this unit covers the basic aspects of E-R modeling. The unit defines the concept of entities, attributes and relationships. Further, it defines different types of entities like strong and weak entities; different types of attributes such as simple, composite, derived etc.; the cardinality and participation constraints in a relationship. These concepts are very useful and you should attempt solving related problems from the further readings. Concepts of EER diagrams including generalisation, specialisation, union

etc. have also been explained in this unit. You may refer to further readings for more details on E-R and EER diagrams.

3.8 SOLUTIONS/ ANSWERS

Check Your Progress 1

1.

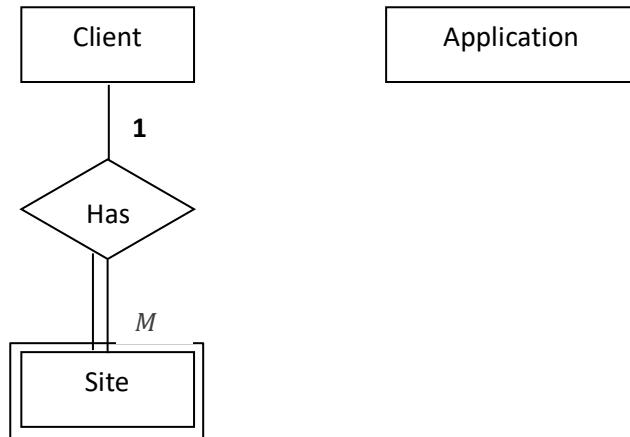
Let us show the step-by-step process of development of Entity-Relationship Diagram for the Client Application system. The first two important entities of the system are **Client** and **Application**. Each of these terms represents a noun, thus, they are eligible to be the entities in the database. But are they the correct entity sets? Client and Application both are **independent** of anything else and the company plans to keep track of its clients and the applications being developed for them. Therefore, each of the entities-Client and Application form an entity set.

But how are these entities related? Are there more entities in the system? Let us first consider the relationship between these two entities, if any. Obviously, the relationship among the entities depends on interpretation of written requirements. Thus, we need to define the terms in more detail.

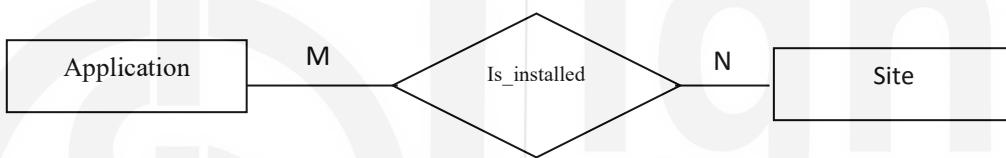
Let us first define the term **Application**. Some of the questions that are to be answered in this regard are: Is keeping track of **Accounts** an application? Is the accounting system installed at each client site regarded as a different application? Can the same application be installed more than once at a particular client site?

Before you answer these questions, do you notice that another entity is in the offering? The **client site** seems to be another candidate entity. This is the kind of thing you need to be sensitive to at this stage of the development of the entity relationship modeling process. So, let us first deal with the relationship between Client and Site before coming back to Application. Just a word of advice: “It is often easier to tackle what seems likely to prove simple before trying to resolve the apparently complex.”

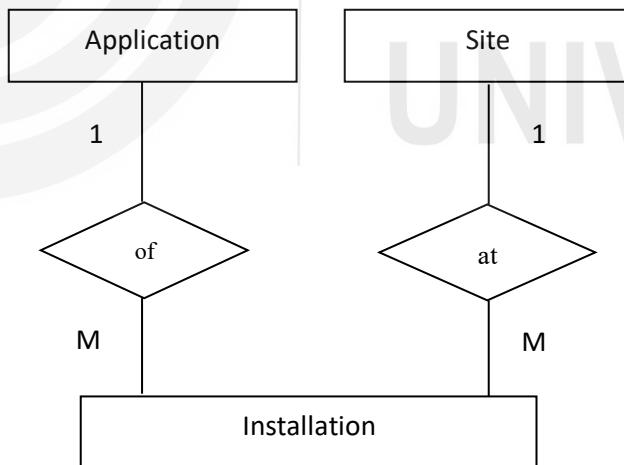
Each Client can have many sites, but each site belongs to one and only one client. Now the question arises what entity type is Site? You cannot have a site without a client. If any site exists without a client, then who would pay the company? This is a good example of an existential dependency and a one-to-many relationship. Thus, Site is a weak entity. This is illustrated in the part E-R diagram given below:



Let us now relate the entity Application to other entities. Please note that several applications developed by the company can be installed at several client sites. Thus, there exists a many-to-many relationship between the entities Site and Application:



However, the M: M relationship “is_installed” has many attributes that need to be stored with it, specifically relating to the authorised persons, setup constraints, dates, etc. Thus, it may be a good idea to promote this relationship as an Entity.



The Figure given above consists of the following relationships:

- of** - relationship describing the installation of applications and
- at** - relationship describing the installation at sites.

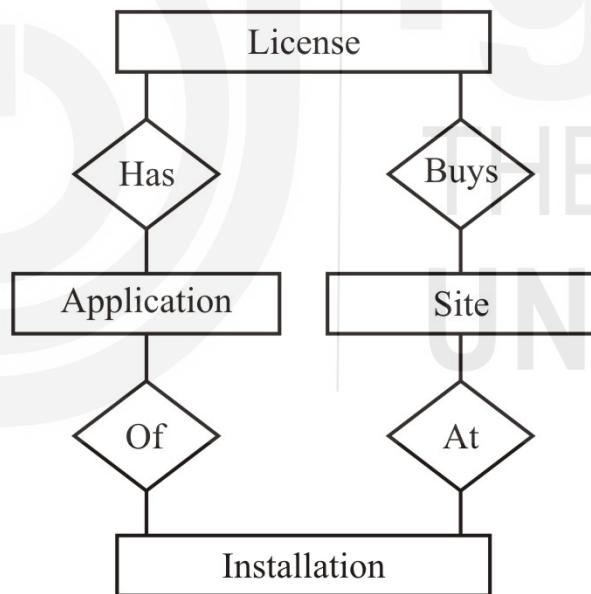
Please note that entities can be recognised in one of two ways – from the nouns of the requirement specification of the system or because of resolving a M:M relationship, as in this case. When you create a new entity in this way, you must find a suitable name for it. This can sometimes be based on the **verbs** used to describe

the M:M. For example, from the statement **you can install the application at many sites**, you can choose the verb install and convert it to related noun **Installation**. But what how will you identify the attributes and relationships of the Installation entity? To find these, you may like to answer the following questions:

- How do you desire to identify each Installation entity?
- What data would be stored in the Installation entity?
- Is an Installation independent of any other entity, that is, can an entity Installation exist without being associated with the entities Client, Site and Application?

In the present design, there cannot be an Installation until you can specify the Client, Site and Application. But since Site is existentially dependent on Client or in other words, Site is subordinate to Client, the Installation can be identified by (Client) Site (it means Client or Site) and Application. You do not want to allow more than one record for the same site and application.

But what if we also sell multiple copies of packages for a site? In such a case, you need to keep track of each individual copy (license number) at each site. In that case, you need another entity named Package (with license number). You may even need separate entities for Application and Package. This will depend on what attributes you want to keep in each of these entities. Thus, with these requirements, the E-R diagram may be extended to as shown below:



Let us define the additional relationships given above:

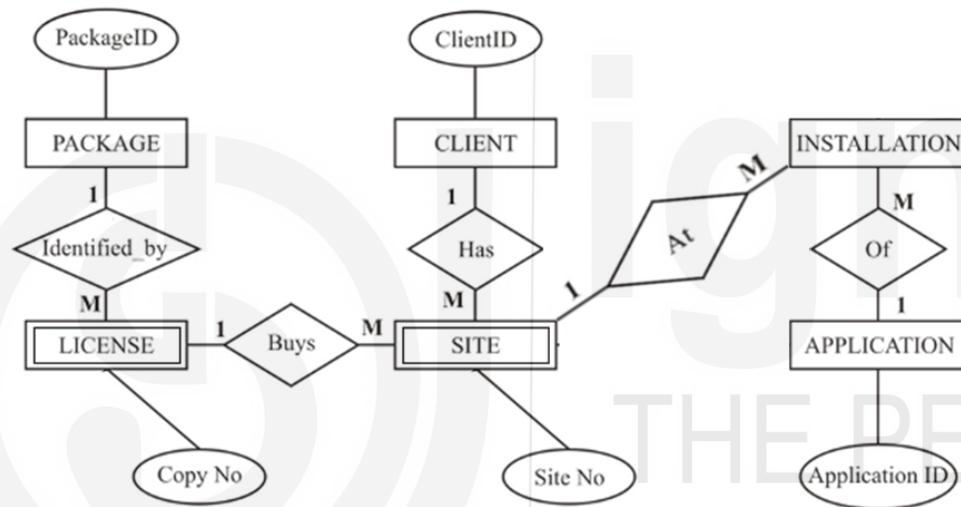
Has: describes that each application has one or more licenses

Buys: describes each site buys the licensed copies of application

You might decide that License should be sub-class to Package entity, therefore, the best unique identifier for the License entity could be Package ID and License serial number. The reason for this relationship is that the license numbers are issued by the companies to whom the Package belongs. The present company does not issue license numbers, and thus have no control over their duration, other service requirements, as

well as the length of data types used for different attributes of the Package and License entity sets. Also, the company does not have control over their uniqueness and changeability of license numbers. Please note that **it is safer to base primary keys on internally assigned values**. What if you make License as a sub-class to Package entity set? It also seems that the client site is not an essential part of the identifier, as the client is free to move the copy of a package to another site. Thus, we should definitely not make client/site part of the primary key of License.

A final proposed E-R diagram for the problem is given below. Please keep thinking and refining your reasoning. Please note that knowing and thinking about a system is essential for making good E-R diagrams. (Please note that in this E-R Diagram, Site and License are modeled as weak entities, though you can decide to change it.)

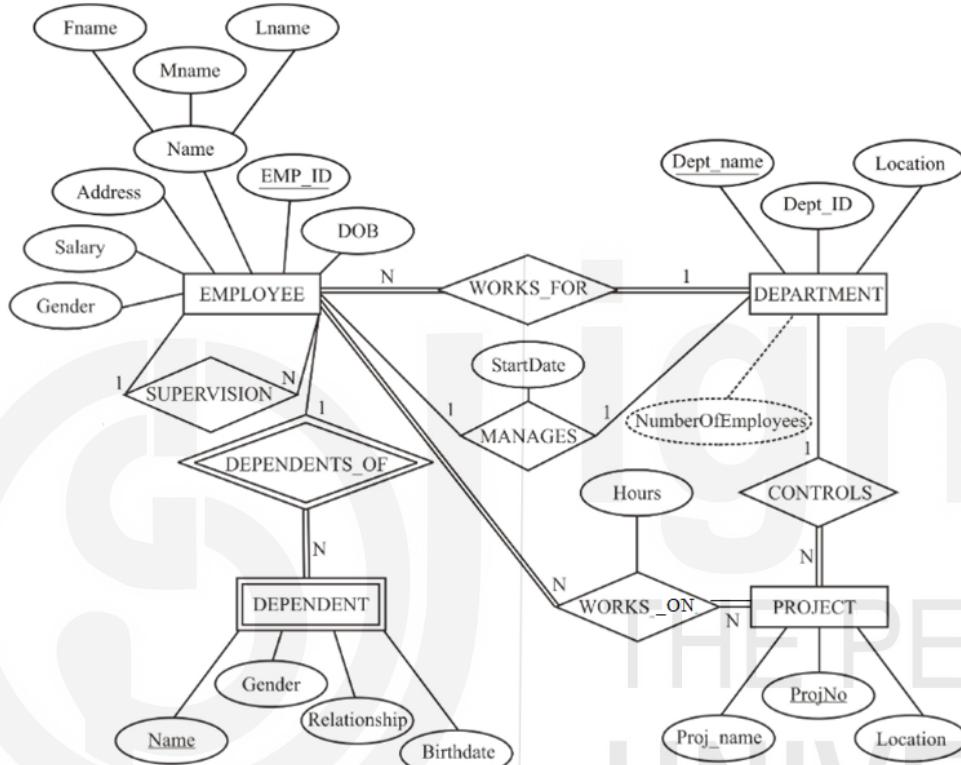


The following table lists probable entities identified so far, together with its superclass, if any, primary keys, and any foreign keys.

Entity	Super Class (if any)	Primary Key	Foreign Keys
Client	-	Client ID	
Site	Client	Client ID, Site No	Client ID
Application	-	Application ID	
Package	-	Package ID	
Installation	Site, Application	Client ID, Site No, Application ID	Client ID, Site No, Application ID

License	Package	Package ID, Copy No	Package ID
---------	---------	---------------------	------------

2) The E-R diagram is given below:



In the E-R diagram given above, **EMPLOYEE** is an entity, who works for a department, i.e., entity **DEPARTMENT**, thus, **WORKS_FOR** is many-to-one relationship, as many employees work for one department. Only one employee (i.e., Manager) manages the department, thus **manages** is the one-to-one relationship.

The attribute **Emp_Id** is the primary key for the entity **EMPLOYEE**, thus **Emp_Id** is unique and NOT NULL. The candidate keys for the entity **DEPARTMENT** are

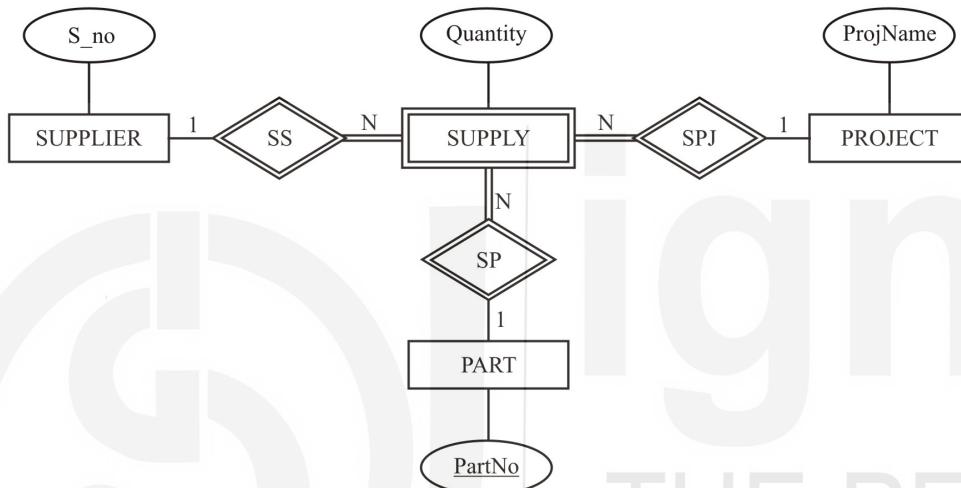
Dept_name and **Dept_ID**. Along with other attributes, **NumberOfEmployees** is the derived attribute on the entity **DEPARTMENT**, which is the number of employees working for a department. Both the entities **EMPLOYEE** and **DEPARTMENT** participate totally in the relationship **WORKS_FOR**, as at least one employee works for the department, similarly an employee must work in a department.

The entity **EMPLOYEES** and the entity **PROJECTS** are related though the many-to many relationship **WORKS_ON**, as many employees can work for one or more than one projects simultaneously. The entity **DEPARTMENT** and entity **PROJECT** has a relationship **CONTROLS**. Since one department controls many projects, thus,

CONTROLS in a 1:N relationship. The entity EMPLOYEE participates totally in the relationship WORKS_ON, as each employee works on at least one project. A project should also have at least one employee, therefore, its participation is also total in WORKS_ON.

The employees can have many dependents, but the entity DEPENDENTS cannot exist without the existence of the entity EMPLOYEE, thus, DEPENDENT is a weak entity. You can very well see the primary keys for all the entities. The underlined attributes in the eclipses represent the primary key.

3. The E-R diagram for supplier-and-parts database is given as follows:



4. Let us first make a simple relation for the E-R diagram in the answer to question 2:

EMPLOYEE(EMP_ID, Fname, Mname, Lname, DOB, Address, Salary, Gender)
DEPARTMENT(Dept_ID, Dept_name, Location)
DEPENDENT(EMP_ID, Name, Gender, Relationship, Birthdate)

Foreign Key: EMP_ID references EMPLOYEE
PROJECT(ProjNO, Proj_name, Location)

WORKS_FOR: due to this relationship the Primary key of 1 side (Dept_ID) will be added to the EMPLOYEE relation.

SUPERVISION: this relationship is on the same entity, therefore, an attribute Supervisor_ID whose domain is EMP_ID will be added to the EMPLOYEE

MANAGES: It is a 1 : 1 relation, you can choose the Department side as there will be less records. It also has an attribute StartDate. Therefore,

MANAGER_ID whose domain is EMP_ID and StartDate attribute would be added to DEPARTMENT.

CONTROLS: It is a 1 : N relationship, so the Primary key of 1 side (Dept_ID) will be added to the PROJECT relation.

DEPENDENT_OF: This relation is already included in DEPENDENT relation.

WORKS_ON: It is a many to many (N : N) relation with total participation on both sides, therefore, a separate table will be created for WORKS_ON with primary key of both the participating entities and attributes of this relation (Hours)

Thus, the final relations would be:

EMPLOYEE (EMP_ID, Fname, Mname, Lname, DOB, Address, Salary, Gender, Dept_ID, Supervisor_ID)
 Foreign Key: Dept_ID references DEPARTMENT.
 Domain Constraint: Domain of Supervisor_ID is EMP_ID.

DEPARTMENT (Dept_ID, Dept_name, Location, MANAGER_ID, StartDate)
 Foreign Key: MANAGER_ID references EMP_ID of EMPLOYEE.

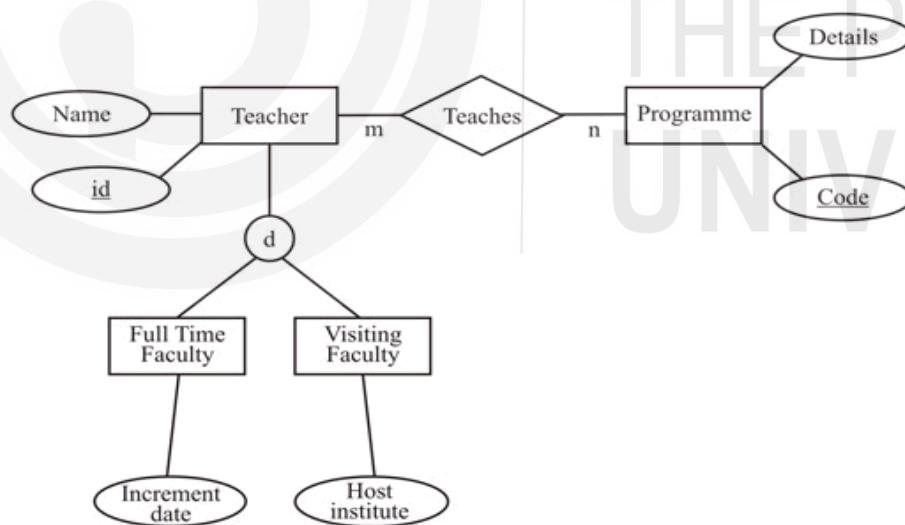
DEPENDENT (EMP_ID, Name, Gender, Relationship, Birthdate)
 Foreign Key: EMP_ID references EMPLOYEE

PROJECT (ProjNo, Proj_name, Location, Dept_ID)
 Foreign Key: Dept_ID references DEPARTMENT.

WORKS_ON (EMP_ID, ProjNo, Hours)

Check Your Progress 2

- 1) The EER diagrams are used to model advanced data models requiring inheritance, specialisation and generalisation.
- 2) The basic constraints used in EER diagrams are disjointness, overlapping and unions.
- 3) For disjointness and union constraints the chances are that you create separate relations for the subclasses and no relation for super class. For overlapping constraints, it is advisable to create a relation of super class and the relations of sub-classes will have only those attributes that are not common to super class except for the primary key.
- 4) The modified EER diagram is:



FullTimeFaculty (id, Name, Increment-date)
VisitingFaculty (id, Name, Host-institute)
Teachers (id, code)
Programme (code, details)

UNIT 4 FILE ORGANISATION IN DBMS

Structure	Page Nos.
4.0 Introduction	
4.1 Objectives	
4.2 Physical Database Design	
4.3 Database Storage on HDD and SSD	
4.4 File Organisations	
4.4.1 Unordered Heap File Organisation	
4.4.2 Sequential File Organisation	
4.4.3 Indexed (Indexed Sequential) File Organisation	
4.4.4 Hashed File Organisation	
4.5 Indexes	
4.6 Implementing Index Using Tree Structure	
4.7 Multi-key File Organisations	
4.7.1 Multiple Access Paths	
4.7.2 Multi-list File Organisation	
4.7.3 Inverted File Organisation	
4.8 Importance of File Organisation in Databases	
4.9 Summary	
4.10 Solutions/Answers	

4.0 INTRODUCTION

In earlier units, you studied the basic concepts of database management systems, entity relationship diagram and relational algebra. Databases are used to store information. Normally, the principal operations you need to perform on database are those relating to:

- Creation of data
- Retrieving the data using conditions
- Modifying
- Deleting some information, which we are sure is no longer useful or valid.

Database structures data as two-dimensional tables, which allows easy processing of these operations. However, as the size of databases are large, the databases are required to be stored on secondary memory of computers (such as hard disk or SSD). Therefore, the secondary storage systems of databases are mainly concerned with the following issues:

- Storing table or tables as files: A single table can be stored as a file or several tables can be put together as a cluster of related records called cluster file.
- Attributes of a table: In general, a table represents the data of one object, therefore, the attributes represent data of a specific object and may be required to be accessed by a specific operation. Thus, it may be a good idea to store all the attributes of an object together. Does the order of attributes in a file matter?
- It seems logical to store all the records of a table contiguously. But, how should such records be ordered? The ordering of records in primary storage does not matter, however, for the secondary storage a specific sequence may be desired. Such decisions may be taken by the database demonstrator and may relate to the performance of the database.
- In the cases of analytical queries, particular attributes are stored together, this approach is called column-oriented approach, this approach is beyond the scope of this Unit. You may refer to further readings for this approach.

This unit focuses on the file Organisation in DBMS, the access methods available and the system parameters associated with them. File Organisation is the way the files are arranged on the disk and access method is how the data can be retrieved based on the file organisation.

4.1 OBJECTIVES

After going through this unit, you should be able to:

- define storage of databases on hard disks and SSD;
- discuss the implementation of various file Organisation techniques;
- discuss the advantages and the limitation of the various file Organisation techniques;
- describe various indexes used in database systems, and
- define the multi-key file organisation.

4.2 PHYSICAL DATABASE DESIGN

The database design involves the process of logical design with the help of E-R diagram, normalisation, etc., followed by the physical design.

The Key issues in the Physical Database Design are:

- The purpose of physical database design is to translate the logical description of data into the technical specifications for storing and retrieving data for the DBMS.
- The goal is to create a design for storing data that will provide adequate performance and ensure database integrity, security and recoverability.

Some of the basic inputs required for Physical Database Design are:

- Normalised relations
- Attribute definitions
- Data usage: entered, retrieved, deleted, updated
- Requirements for security, backup, recovery, retention, integrity
- DBMS characteristics.
- Performance criteria such as response time requirement with respect to volume estimates.

However, for such data some of the Physical Database Design Decisions that are to be taken are:

- Optimising attribute data types.
- Modifying the logical design.
- Specifying the file Organisation.
- Choosing indexes.

Designing the attributes in the database

The following are the considerations one has to keep in mind while designing the attributes in the database.

- Choosing data type
- Coding, compression, encryption
- Controlling data integrity
- Default value
 - Range control
 - Null value control
 - Referential integrity
- Handling missing data

- Substitute an estimate of the missing value
- Trigger a report listing missing values
- In programs, ignore missing data unless the value is significant.

Physical Records

These are the records that are stored in the secondary storage devices. For a database relation, physical records are the group of fields stored in adjacent memory locations and retrieved together as a unit. Considering the page memory system, a data page is the amount of data read or written in one I/O operation to and from a secondary storage device to the memory and vice-versa. In this context we define a term blocking factor that is defined as the number of physical records per page.

The issues relating to the Design of the Physical Database Files

Physical File is a file as stored on the disk or SSD. The main issues relating to physical files are:

- Constructs to link two pieces of data:
 - Sequential storage.
 - Pointers.
- File Organisation: How are the files arranged on the disk?
- Access Method: How can the data be retrieved based on the file Organisation?

Let us see in the next section how the data is stored on the hard disks (HDD) or SSDs

4.3 DATABASE STORAGE ON HDD OR SSD

At this point, it is worthwhile to note the difference between the terms file Organisation and the access method. A file organisation refers to the organisation of the data records, which are part of a file, into a group of records that are placed in a block of secondary storage. The file organisation also entails the access structures and interlinking of records. An access method is the way - how the data can be retrieved based on the file Organisation.

Mostly the databases are stored persistently on HDD or SSD for the reasons given below:

- The databases being very large may not fit completely in the main memory.
- Data of the database is to be stored permanently using non-volatile storage.
- Primary storage is expensive, using secondary storage reduces the cost of the storage per unit.

Each hard drive is usually composed of a set of disk platters. Each disk platter has a layer of magnetic material deposited on its surface. The entire disk can contain a large amount of data, which is organised into smaller packages called BLOCKS (or pages). On most computers, one block is equivalent to 1 KB of data (= 1024 Bytes). A block is the smallest unit of data transfer between the hard disk and the processor of the computer. Each block therefore has a fixed, assigned, address. Typically, the computer processor will submit a read/write request, which includes the address of the block, and the address of RAM in the computer memory area called a buffer (or cache) where the data must be stored/taken from. The processor then reads and modifies the buffer data as required and, if required, writes the block back to the disk. Let us see how the tables of the database are stored on the hard disk.

A Solid-State Disk (SSD) is made up of flash memories. These SSD devices also provide similar block-oriented access to data, however, since they do not have physically moving components, they provide lower latency and lower access time than that of disks. Therefore, in the subsequent sections, we will provide details on the disk oriented approach of data storage.

Fixed and Variable Length Records

There are two basic ways of storing a record on the disks – Fixed Length records and Variable Length Records. In the fixed length records all the attributes are assigned equal space in terms of bytes, just like fixed length structure in C programming. Thus, each record will be of the same size. In such cases, only metadata can be used to identify different records and their attributes.

As far as variable length records are concerned, it may be noted that each record of a table may be of different length. This is because in some records, some attribute values may be 'null'; or some attributes may be of type *varchar*, which allows variable number of characters to be stored in an attribute, therefore each record may have a different length string as the value of this attribute. To store variable length records, it is necessary to use a character that marks the end of an attribute value. In addition, an end of record marker will also be needed, as one disk block may store several records. Therefore, each record is separated from the next, again by another special character, the record separator.

The next section discusses different types of file organisation that can be used to store the files on the disks.

4.4 FILE ORGANISATIONS

File organisation is used to organise the content of a file on a secondary storage device. A good file organisation should support efficient data access and update operations on the content of a file, which is stored on the secondary storage.

Data files are organised so as to facilitate access to records and to ensure their efficient storage. A tradeoff between these two requirements generally exists: if rapid access is required, more storage space is required to make it possible. Selection of File Organisations is dependent on two factors, as shown below:

- Typical DBMS applications may need to access a small subset of the data of a database at any given time.
- Whenever a portion of the data is needed by the DBMS; it is located on disk, copied to memory for processing, and rewritten to disk if the data was modified.

A file of record is likely to be accessed and modified in a variety of ways, and different ways of arranging the records enable different operations over the file to be carried out efficiently. A DBMS supports several file Organisation techniques. The important task of the DBA is to choose a good Organisation for each file based on its type of use.

For a database system, you select a file organisation a suitable file organisation based on the parameters like number of keys used to access the files, the ratio of update and access operations, the type of storage technology etc. *Figure 4.1* uses access keys as the basis for classifying different file organisations. This classification will be used in this unit to describe various file organisations.

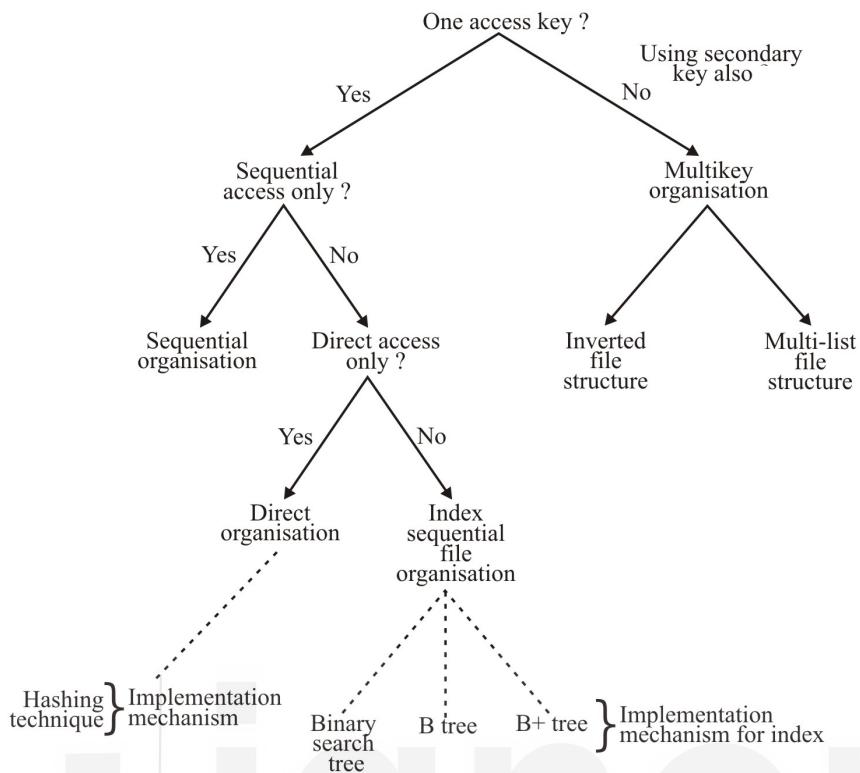


Figure 4.1: Different File Organisation Based on Access Keys

Let us discuss some of these techniques in more detail:

4.4.1 Unordered Heap File Organisation

Basically, these files are unordered files. It is the simplest and most basic type. These files consist of randomly ordered records. The records will have no particular ordering. The operations that you can perform on the records of a heap file are insert, retrieve and delete. The features of the heap file Organisation are:

- New records can be inserted in any empty space that can accommodate them.
- When old records are deleted, the occupied space becomes empty and available for any new insertion.
- If updated records grow, they may need to be relocated (moved) to a new empty space. Thus, this file organisation keeps a list of empty spaces.

Advantages of heap files

1. This is a simple file Organisation.
2. Insertion is somehow efficient.
3. Good for bulk-loading data into a table.
4. Best if file scans are common or insertions are frequent.

Disadvantages of heap files

1. Retrieval requires a linear search and is inefficient.
2. Deletion can result in unused space/need for reorganisation.

4.4.2 Sequential File Organisation

In the sequential Organisation, records of the file are stored in sequence (or consecutively) by the primary key field values. In this organisation, the records can be accessed in the order of its primary key. This kind of file Organisation works well for tasks in which nearly every record of the file is required to be accessed, such as the payroll system. Figure 4.2 shows this file organisation.

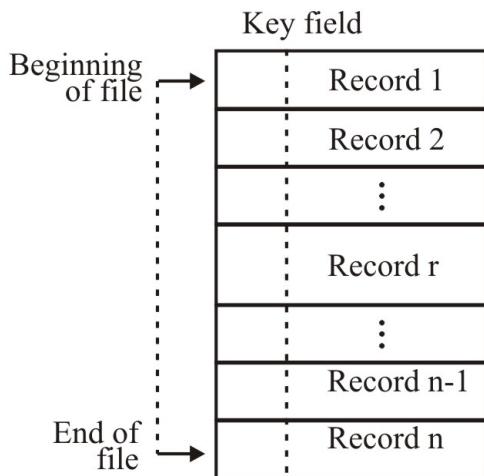


Figure 4.2: Structure of sequential file

A sequential file maintains the records in the logical sequence of its primary key values. Sequential files are inefficient for random access, however, are suitable for sequential access. A sequential file can be stored on devices like magnetic tape that allow sequential access.

On an average, to search a record in a sequential file would require looking into half of the records of the file. However, if a sequential file is stored on a disk (remember disks support direct access of its blocks) with keys stored separately from the rest of the record, then only those disk blocks are needed to be read that contains the desired record or records. This type of storage allows binary search on sequential file blocks, thus, enhancing the speed of access.

Updating a sequential file usually creates a new file so that the record sequence on the primary key is maintained. The update operation first copies the records till the record after which update is required into the new file and then the updated record is put followed by the remainder of records. Thus, the method of updating a sequential file automatically creates a backup copy. However, such update operations are very time consuming.

Addition of records in the sequential files are also handled in a similar manner to update operation. If a record is to be inserted at the last record of the file, it can be performed very easily. However, if a record is required to be inserted in between two records, then such insertion would require shifting down all the subsequent records in the file by one record space. In case of deletion of a record, all the subsequent records need to be shifted up by one record space.

Sequential file organisation is most suitable, if all the records of a file are to be processed in a sequence. For example, processing the monthly payroll of all the employees of an organisation, will require processing of all the employees records sequentially. However, a single update is expensive as a new file must be created, therefore, to reduce the cost per update, all update requests are stored in a single update file, which is sorted in the order of the sequential file ordering key. The file containing the updates is sometimes referred to as a transaction file and is used to update the sequential file in a single processing cycle.

This process is called the batch mode of updating. In this mode, each record of the master sequential file is checked for one or more possible updates by comparing with the update information of the transaction file. The records are written to a new master file in a sequential manner. A record that requires multiple updates is written only when all the updates have been performed on the record. A record that is to be deleted

is not written to a new master file. Thus, a new updated master file will be created from the transaction file and the old master file.

Thus, update, insertion and deletion of records in a sequential file require a new file creation. Can we reduce creation of this new file? Yes, it can be done easily, if the original sequential file is created with holes, which are empty record spaces, as shown in the *Figure 4.3*. Thus, reorganisation of file on addition and update operation can be restricted to only one block, which is read into/ written from the main memory as a single unit. Thus, holes increase the performance of sequential file insertion and deletion. This organisation also supports a concept of overflow area, which can store the spilled over records if a block is full. This technique is also used in index sequential file organisation. A detailed discussion on it can be found in the further readings.

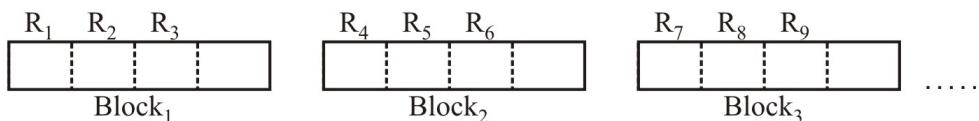


Figure 4.3: A sequential file with empty spaces for record insertions

Advantages of Sequential File Organisation

- It is fast and efficient when dealing with large volumes of data that need to be processed periodically (batch system).

Disadvantages of sequential File Organisation

- Requires all the new transactions to be sorted into a proper sequence for sequential access processing.
- Locating, storing, modifying, deleting, or adding records in the file requires rearranging the file.
- This method is too slow to handle applications requiring immediate updating or responses.

4.4.3 Indexed (Indexed Sequential) File Organisation

It organises the file like a large dictionary, i.e., records are stored in order of the key, but an index is kept which also permits a type of direct access. The records are stored sequentially by primary key values and there is an index built over the primary key field.

To locate a record in a sequential file, which is not indexed sequential, on average, you may read about half the number of records. Therefore, retrieval of a specific record in a sequential file is inefficient, especially if the file has a very large number of disk storage blocks. The use of index in a sequential file improves the query response time by adding an index, which is defined as a set of *<index value, address>* pair. An index is a mechanism for faster search, as the size of the index is much smaller than the original file. Thus, an index may be contained entirely in the main memory of the computer.

An indexed sequential file is a sequential file, which is stored in the order of its primary key, that contains an index on its primary key. Indexed sequential file allows advantages of indexing and sequential organization of records. The sequential organisation facilitates sequential processing of records, whereas the index helps in enhancing the performance of locating specific records based on the index value. In order, to make such files more efficient for insertion and deleting of records, such files are supported with overflow blocks. This reduces the need of moving all the records in a file. Figure 4.6 is an example of an indexed sequential file. Please note that the index is represented in Figure 4.6 as *<Primary Index Value, Block Pointer>*.

Hashing is the most common form of purely random access to a file or database. It is also used as an optimisation technique to access columns that do not have an index. Hashing involves the use of a hash function. Input to a hash function is the value of the attribute or set of attributes of a record that are to be used for file organisation and the output is the block address or page address, where that record can be found. Figure 4.4 shows a file using hashing file organisation. The hash function used for this file is $\text{key mod } 4$. Notice that records with different key values can be placed in a Block based on the hashing function. To search the location of a record, you can apply the hashing function on the key value and then search the hashed block. For example, if you are searching for the location of key 29, then the record can be found in $29 \bmod 4 = \text{Block 1}$, read this Block in the main memory and do a linear search on key value to locate the record in the main memory. The most popular form of hashing is division hashing with chained overflow. You can refer to further readings for more details on this file organisation.

Advantages of Hashed File Organisation

1. Insertion or search on hash-key is fast.
2. Best if equality search is needed on hash-key.

Disadvantages of Hashed File Organisation

1. It is a complex file Organisation method.
2. Search is slow.
3. It suffers from disk space overhead.
4. Unbalanced buckets degrade performance.
5. Range search is slow.



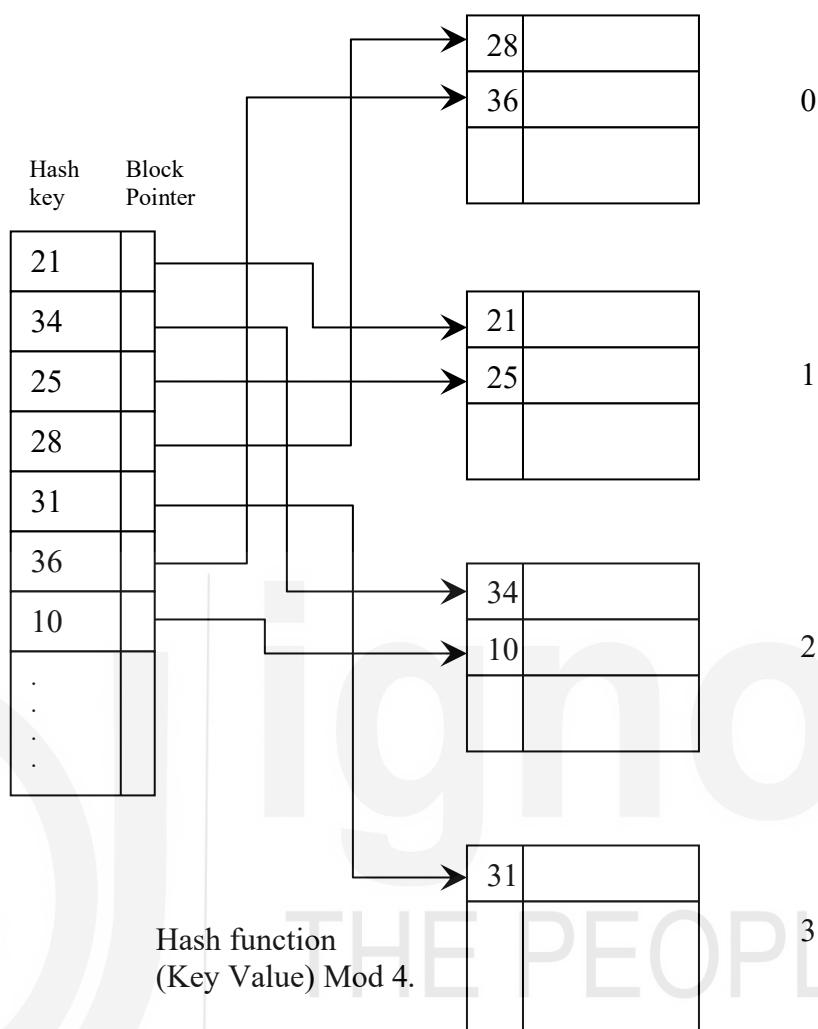


Figure 4.4: A Hashed file

Check Your Progress 1

- 1) List five operations on sequential file Organisation. Comment on the performance of each of these operations.

- 2) What are Direct-Access systems? What can be the various strategies to achieve this?

- 3) What is file organisation and what are the essential factors that are to be considered for a file organisation?

4.5 INDEXES

One of the terms used during the file organisation is the term index. In this section, let us define this term in more detail.

Every printed book that you read, in general, has an index of keywords at the end. Notice that this index is a sorted list of keywords (index values) and page numbers (address) where the keyword can be found. In databases also an index is defined in a similar way, as the \langle index value, address \rangle pair.

The basic advantage of having sorted index pages at the end of the book is that you can locate the description about a desired keyword in the book. You could have used the topic and subtopic listed in the table of contents, but it is not necessary that the given keyword can be found there; also, they are not in any sorted sequence. If a keyword is not listed in index and table of contents, then you need to search each page of the book to find the required keyword, which is very cumbersome. Thus, an index at the back of the book helps in locating the required keyword references very easily in the book.

The same is true for databases that have a very large number of records. A database index allows fast search on the index value in database records. It will be difficult to locate an attribute value in a large database, if the index on that attribute is not provided. In such a case the value is to be searched record-by-record in the entire database, which is cumbersome and time consuming. It is important to note that for a large database all the records cannot be kept in the main memory at a time, thus, data needs to be transferred from the secondary storage device, which is more time consuming.

An index entry consists of a pair consisting of index value and a list of pointers to disk blocks for the records that have that index value. An index contains such information for every stored value of the index attribute. An index file is very small compared to a data file that stores a relation. Also index entries are ordered, so that an index can be searched using an efficient search method like binary search. In case an index file is very large, you can create a multi-level index, that is index on index. Multi-level indexes are defined later in this section.

There are many types of indexes that are categorised as:

Primary index	Single level index	Spare index
Secondary index	Multi-level index	Dense index
Clustering index		

A primary index is defined on the attributes *in the order of which the file is stored*. This field is called the ordering field. A primary index can be on the primary or candidate key of a file. If an index is on the ordering attributes, which are not candidate key attributes, then several records may be related to one ordering field value. This is called clustering index. It is to be noted that there can be only one physical ordering attribute or set of attributes for a file. Thus, a file can have either the primary index or clustering index, not both. Secondary indexes are defined on the non-ordering fields. Thus, there can be several secondary indexes in a file, but only one primary or clustering index.

Primary index

Primary index is a file that contains a sorted sequence of index records having two columns: the ordering key field; and a block address for that key field in the data file. The ordering key field for this index can be the primary key of the data file. Primary index contains one index entry of the ordering key field for each Block of data. An entry in the primary index file contains either the key value of the first record or the key value of the last record, which are stored in that data block; and a pointer to that data block.

Let us discuss the primary index with the help of an example. Let us assume a student database as (Assuming that one block stores only four student records). Figure 4.5 shows a sample of this data file. The sample file is ordered on the attribute - enrolment number.

Block Number	Enrolment Number	Name	City	Programme
BLOCK 1	2109348	...	CHENNAI	CIC
	2109349	...	CALCUTTA	MCA
	2109351	...	KOCHI	BCA
	2109352	...	KOCHI	CIC
BLOCK 2	2109353	...	VARANASI	MBA
	2238389	...	NEW DELHI	MBA
	2238390	...	VARANASI	MCA
	2238411	...	NEW DELHI	BCA
BLOCK 3	2238412	...	AJMER	MCA
	2238414	...	NEW DELHI	MCA
	2238422	...	MUMBAI	BSC
	2258014	...	MUMBAI	BCA
BLOCK 4	2258015	...	MUMBAI	BCA
	2258017	...	NEW DELHI	BSC
	2258018	...	MUMBAI	MCA
	2258019	...	LUCKNOW	MBA
...
BLOCK r	2258616	...	AJMER	BCA
	2258617	...	LUCKNOW	MCA
	2258618	...	NEW DELHI	BSC
	2318935	...	FARIDABAD	MBA
...
BLOCK N-1	2401407	...	BAREILLY	CIC
	2401408	...	BAREILLY	BSC
	2401409	...	AURANGABAD	BCA
	2401623	...	NEW DELHI	MCA
BLOCK N	2401666	...	MUMBAI	MCA
	2409216	...	LUCKNOW	MBA
	2409217	...	ALMORA	BCA
	2409422	...	MUMBAI	BSC

Figure 4.5: A Student file stored in the order of student enrolment numbers

The primary index on this file would be on the ordering field – enrolment number. The primary index on this file is shown in Figure 4.6. Please note the following points in Figure 4.6.

- An index entry is defined as the attribute value, pointer to the block where that record is stored. The pointer physically is represented as the binary address of the block.
- Since there are four student records, which of the key values should be stored as the index value? We have used the first key value stored in the block as the

index key value. This is also called the anchor value. All the records stored in the given block have ordering attribute value as the same or more than this anchor value.

- The primary index may be smaller in size, as it contains one index entry for each storage data block. Also notice that not all the records need to have an entry in the index file. This type of index is called a non-dense index. Thus, the primary index is non-dense index.
- To locate the record of a student whose enrolment number is 2238422, you need to find two consecutive entries of indexes such that index value $1 < 2238422 <$ index value 2. In the Figure 4.6, you can find the third and fourth index values as: 2238412 and 2258015 respectively satisfying the properties as above. Thus, the required student record must be found in Block 3.

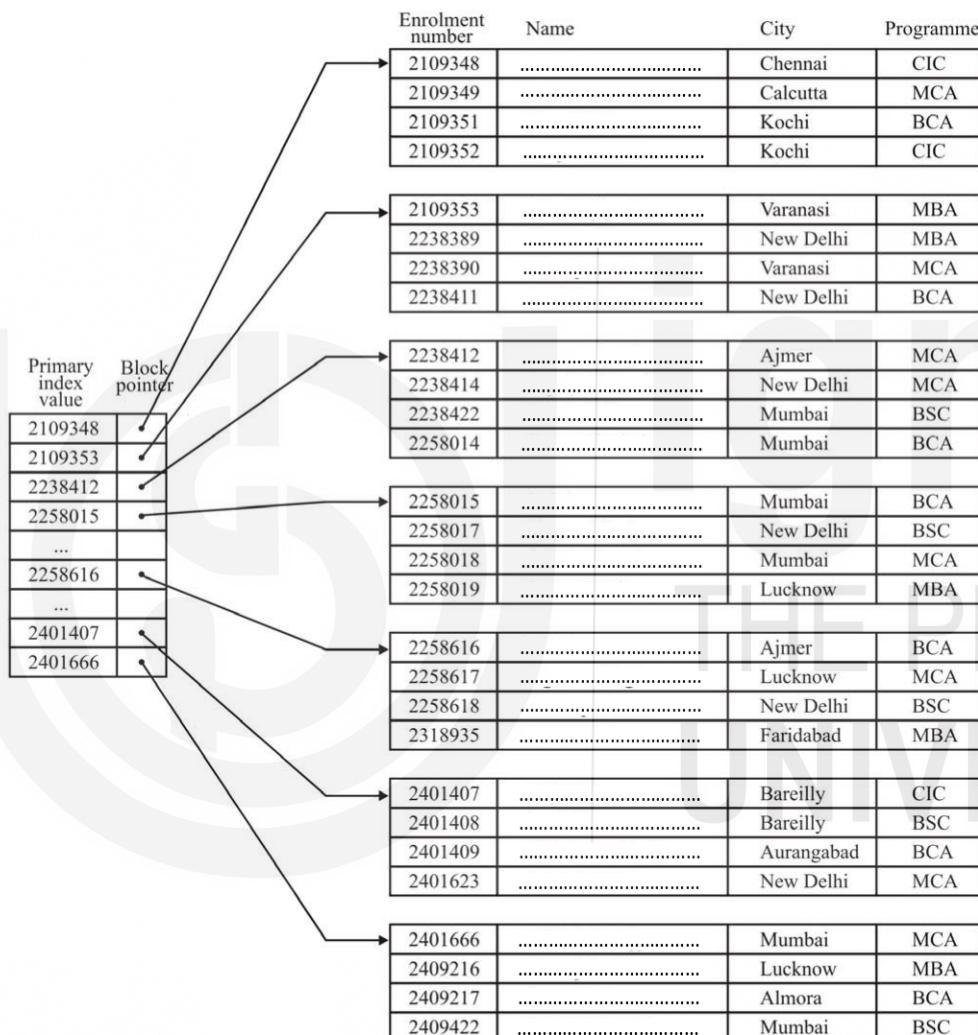


Figure 4.6: A Student file and the Primary Index on Enrolment Number

But does primary index enhance efficiency of searching? Let us explain this with the help of an example (Please note we will define savings in terms of the number of block transfers, as that is the most time-consuming operation during searching).

Example 1: An ordered student file (ordering field is enrolment number) has 20,000 records stored on a disk having the Block size as 1 K. Assume that each student record is of 100 bytes, the ordering field is of 8 bytes, and block pointer is also of 8 bytes, find how many block accesses on average may be saved on using primary index.

Answer:

Number of accesses without using Primary Index:

Number of records in the file = 20000

Block size = 1024 bytes
Record size = 100 bytes
Number of records per block = integer value of $[1024 / 100] = 10$
Number of disk blocks acquired by the file
= [Number of records / records per block]
= $[20000/10] = 2000$
Assuming a block level binary search, it would require $\log_2 2000$
= about 11 block accesses.

Number of accesses with Primary Index:

Size of an index entry = $8+8 = 16$ bytes
Number of index entries that can be stored per block
= integer value of $[1024 / 16] = 64$
Number of index entries = number of disk blocks = 2000
Number of index blocks = ceiling of $[2000/ 64] = 32$
Number of index block transfers to find the value in index blocks = $\log_2 32 = 5$
One block transfer will be required to get the data records using the index pointer after the required index value has been located.
So total number of block transfers with primary index = $5 + 1 = 6$.

Thus, the Primary index would save $11 - 6 = 5$ block transfers for the given size of data and index.

Is there any disadvantage of using a primary index? Yes, a primary index requires the data file to be ordered, this causes problems during insertion and deletion of records in the file. This problem can be taken care of by selecting a suitable file organisation that allows logical ordering only.

Clustering Indexes.

It may be a good idea to keep records of the students in the order of the programme they have registered, as most of the data file accesses may require programme wise student data. A file can be ordered and physically stored on non-key attributes; an index that is created on such non-key attributes would have multiple records pointed to by a single index entry. Such an index is called a clustering index. *Figure 4.7 and Figure 4.8* show the clustering indexes in the same file organised in different ways.

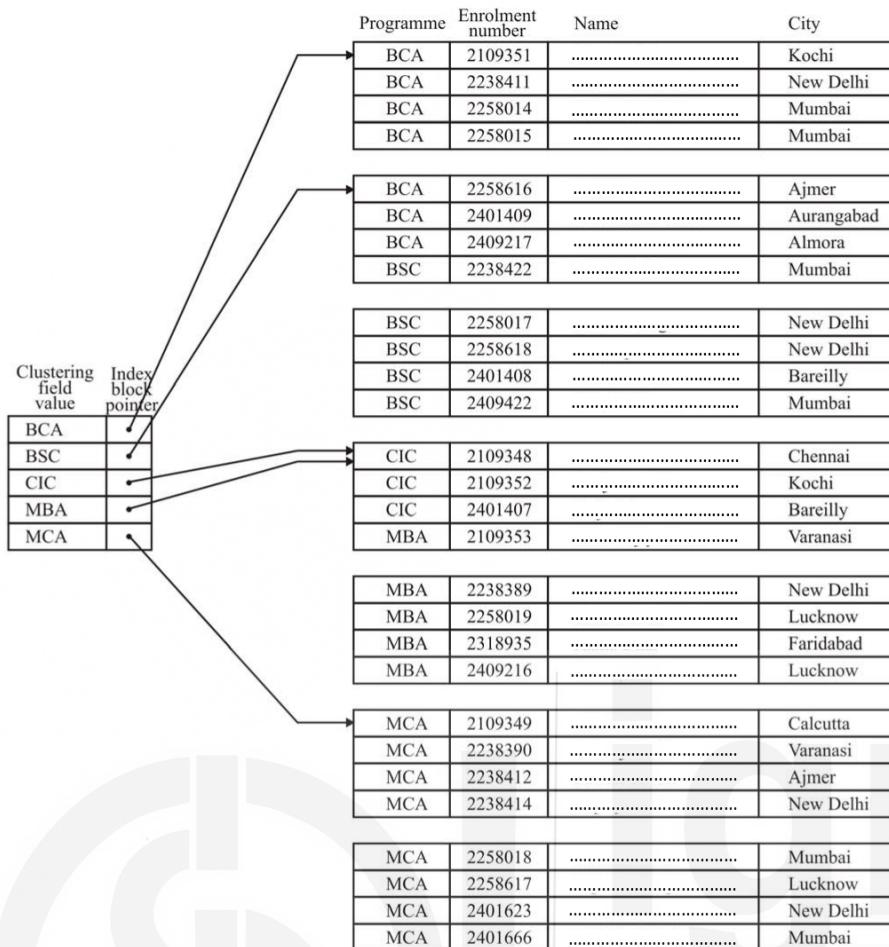


Figure 4.7: A clustering Index on Programme in the Student file

Please note the following points about the clustering index as shown in the *Figure 4.7*:

- The clustering index is an ordered file having the clustering index value and a block pointer to the first block where that clustering field value first appears.
- Clustering index is also a sparse index. The size of the clustering index is smaller than the primary index as far as the number of entries is concerned.

Please note that in Figure 4.7, the data file can have a single block in which data of students of multiple programmes are stored. You can improve upon this organisation by allowing only one Programme data in one block. Such an organisation and its clustering index is shown in the *Figure 4.8*:

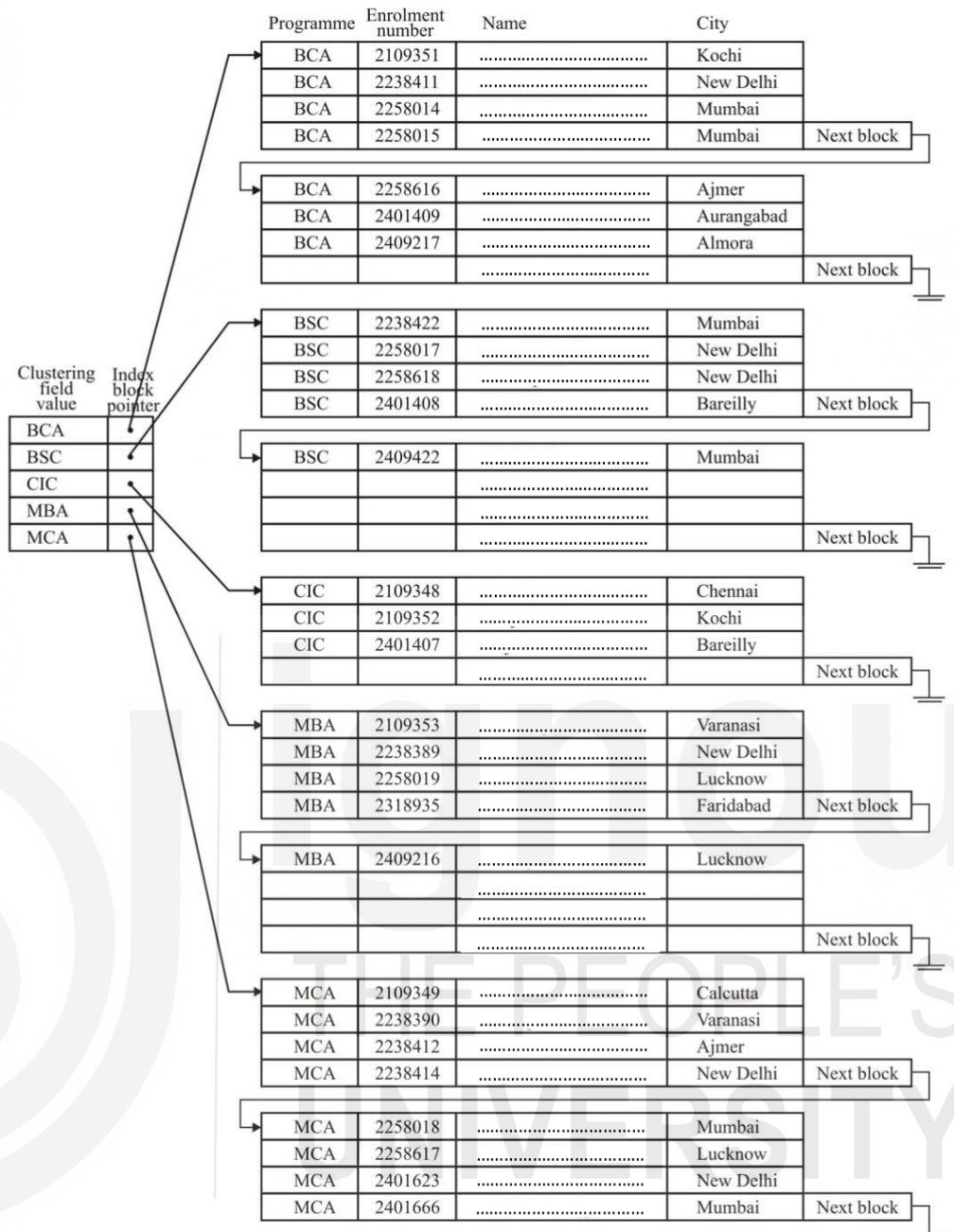


Figure 4.8: Clustering index with separate blocks for each clustering attribute value

Please note the following points for the above:

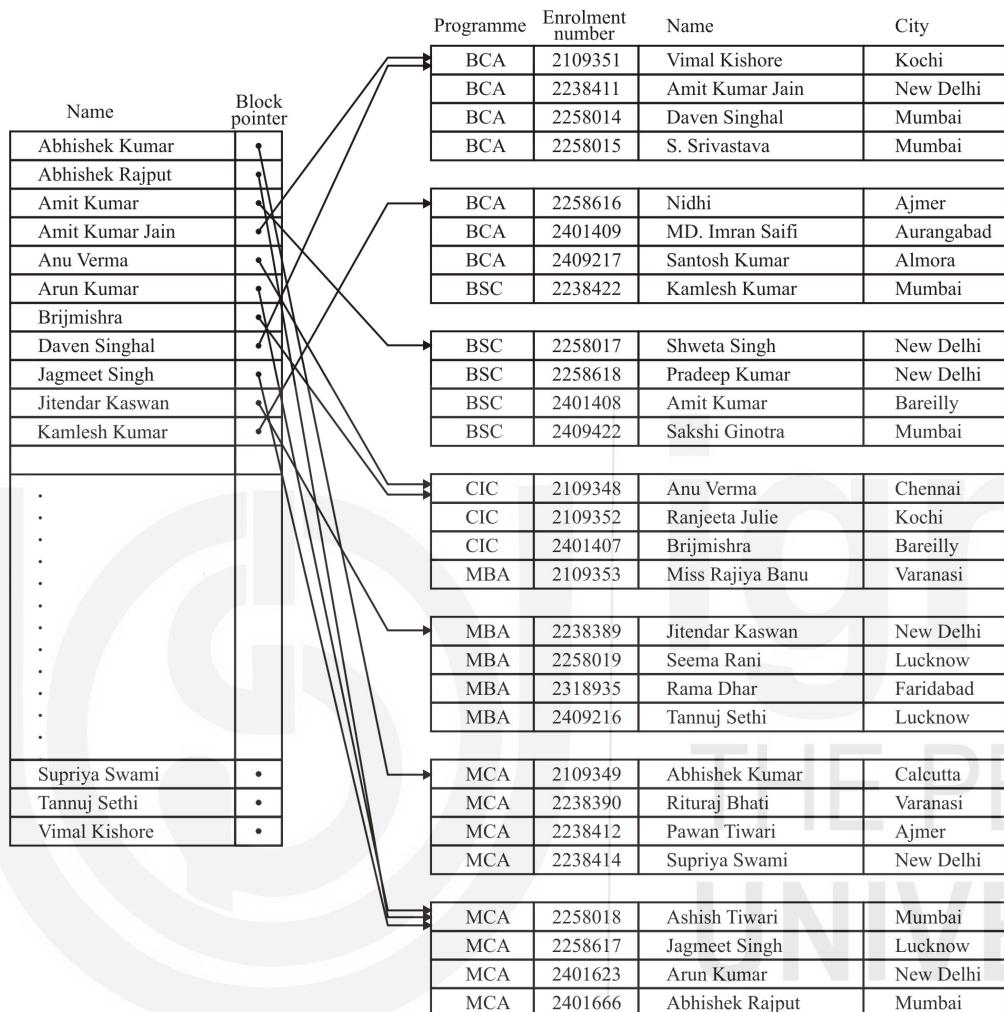
- Data insertion and deletion is easier than in the earlier clustering files, yet even now it is cumbersome.
- The blocks allocated for one index entry are in the form of linked list blocks.
- Clustering index is another example of a non-dense index as it has one entry for every distinct value of the clustering index attribute and not for every record in the file.

Secondary Indexes

Consider the student database and its primary and clustering index (only one will be applicable at a time). Now consider the situation when the database is to be searched or accessed in the alphabetical order of student names. Any search on a student name would require a sequential data file search, as there is no ordering on student names. Thus, searching for a student's name is going to be very time consuming. Such a search on an average would require reading half of the total number of blocks. Thus, you need secondary indices in database systems. A secondary index is a file that contains records containing a secondary index field value which is not the ordering field of the data file, and a pointer to the block that contains the data record. Please

note that although a data file can have only one primary index (as there can be only one ordering of a database file), it can have many secondary indices.

Secondary indexes can be defined on an alternate key or non-key attributes. A secondary index that is defined on the alternate key will be dense, while a secondary index on non-key attributes would require a bucket of pointers for one index entry. Let us explain them in more detail with the help of *Figure 4.9*.



**Figure 4.9: A Dense Secondary Index on a non-ordering key field of a file
(The file is organised on the clustering field “Programme”)**

Please note the following in *Figure 4.9*.

- The names in the data file are unique and thus are being assumed as the alternate key. Each name therefore is appearing as the secondary index entry.
- The pointers are block pointers, thus are pointing to the beginning of the block and not a record. For simplicity, we have not shown all the pointers in Figure 4.9.
- This type of secondary index file is dense index as it contains one entry for each record/distinct value.
- The secondary index is larger than the Primary index as we cannot use block anchor values here as the secondary index attributes are not the ordering attribute of the data file.
- To search a value in a data file using name, first the index file is (binary) searched to determine the block, where the record having the desired key value

can be found. Then this block is transferred to the main memory where the desired record is searched and accessed.

- A secondary index file, usually, has a larger number of index entries than that of primary index. However, the secondary index improves the search time to a greater proportion than that of a primary index. This is due to the reason - If a primary index does not exist even then, you can perform binary search on the blocks of data records, as the records are ordered in the sequence of primary index value. However, if a secondary key does not exist, then you may need to search the records sequentially. This fact is demonstrated with the help of Example 2.

Example 2: Let us reconsider the problem of Example 1 with a few changes. An unordered student file, which is not ordered on a primary key, has 20,000 records stored on a disk having a Block size as 1 K. Assume that each student record is of 100 bytes, the secondary index field is of 8 bytes, and the block pointer is also of 8 bytes, find how many block accesses on average may be saved on using a secondary index on enrolment number.

Answer:

Number of accesses without using a Secondary Index:

Number of records in the file = 20000

Block size = 1024 bytes

Record size = 100 bytes

Number of records per block = integer value of $[1024 / 100] = 10$

Number of disk blocks acquired by the file

$$= [\text{Number of records} / \text{records per block}]$$

$$= [20000/10] = 2000$$

Since the file is un-ordered any search on an average will require about half of the above blocks to be accessed. Thus, average number of block accesses = 1000

Number of accesses with Secondary Index:

Size of an index entry = $8+8 = 16$ bytes

Number of index entries that can be stored per block

$$= \text{integer value of } [1024 / 16] = 64$$

Number of index entries = number of records = 20000

Number of index blocks = ceiling of $[20000 / 64] = 320$

Number of index block transfers to find the value in index blocks

$$= \text{ceiling of } [\log_2 320] = 9$$

One block transfer will be required to get the data records using the index pointer after the required index value has been located. So total number of block transfers with secondary index = $9 + 1 = 10$

Thus, the Secondary index would save about 1990 block transfers for the given case. This is a huge saving compared to a primary index. Please also compare the size of the secondary index to the primary index.

Let us now see an example of a secondary index that is on an attribute that is not an alternate key.

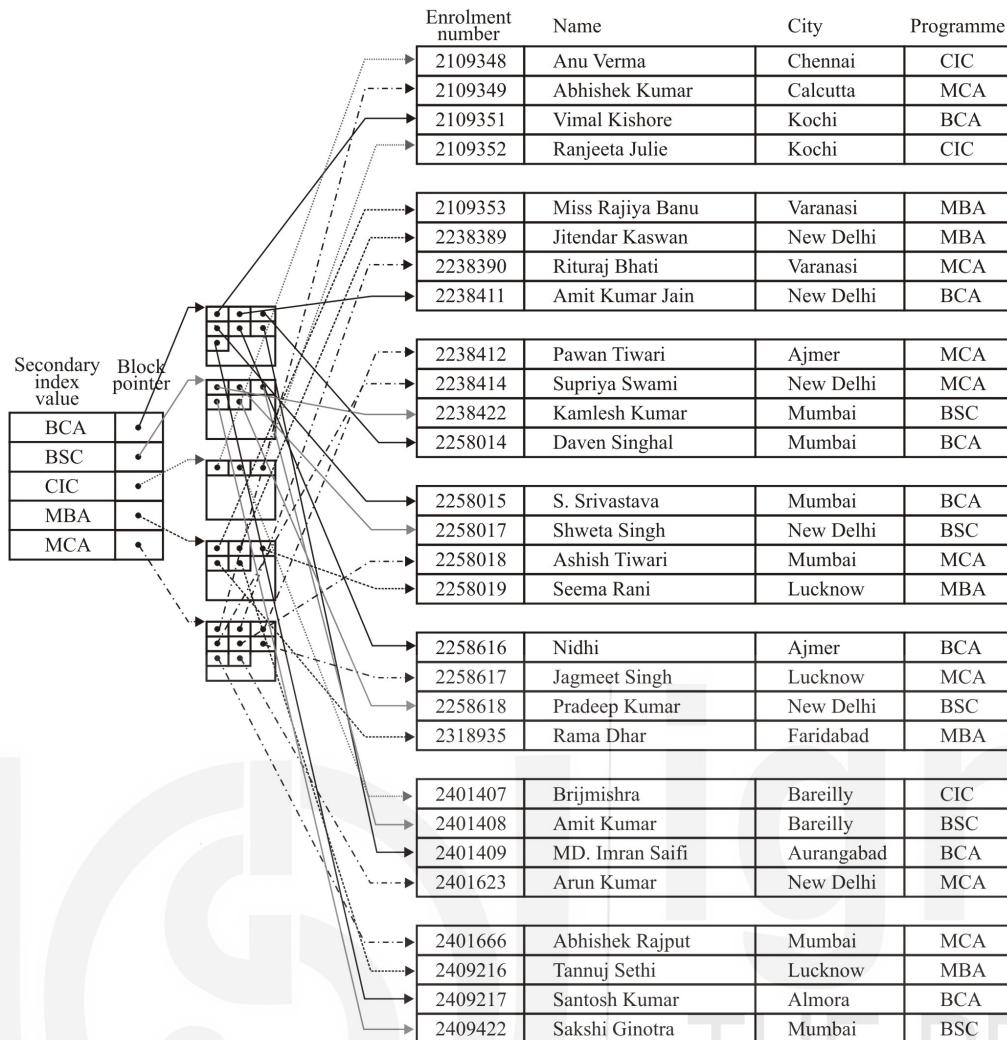


Figure 4.10: An example of Secondary Index with single level of indirection

(Index entries are of fixed length and have unique field values)

(The file is organised on the primary key)

A secondary index that needs to be created on a field that is not a candidate key can be implemented using several ways. We have shown here the way in which a block of pointer records is kept for implementing such an index. This method allows the index entries to be of fixed length. It also allows only a single entry for the value of the indexing attribute. In addition, the level of indirection allows multiple index pointers to be stored in a single block of data. The algorithms for searching the index, inserting and deleting new values into an index are very simple in such a scheme. Thus, this is the most popular scheme for implementing such secondary indexes.

Sparse and Dense Indexes

As discussed earlier, an index is defined as the ordered *<index value, address>* pair. These indexes in principle are the same as that of indexes used at the back of the book. The key ideas of the indexes are:

- They are sorted on the order of the index value (ascending or descending) as per the choice of the creator.
- The indexes are logically separate files (just like separate index pages of the book).
- An index is primarily created for fast access to information.
- The primary index is the index on the ordering field of the data file, whereas a secondary index is the index on any other field, thus, is more useful.

But what are sparse and dense indexes?

A dense index contains one index entry for every value of the indexing attributes, whereas a sparse index also called non-dense index contains few index entries out of the available indexing attribute values. For example, the primary index on enrolment number is sparse, while secondary index on student name is dense.

Multilevel Indexing Scheme

For small files, the indexing scheme keeps the address of the block file in each index entry. Such indices would be small and can be processed efficiently in the main memory. However, for a large file the size of the index can also be very large. In such a case, you can create indexes at several levels, with the last level pointing to the data records. Figure 4.11 shows this scheme.

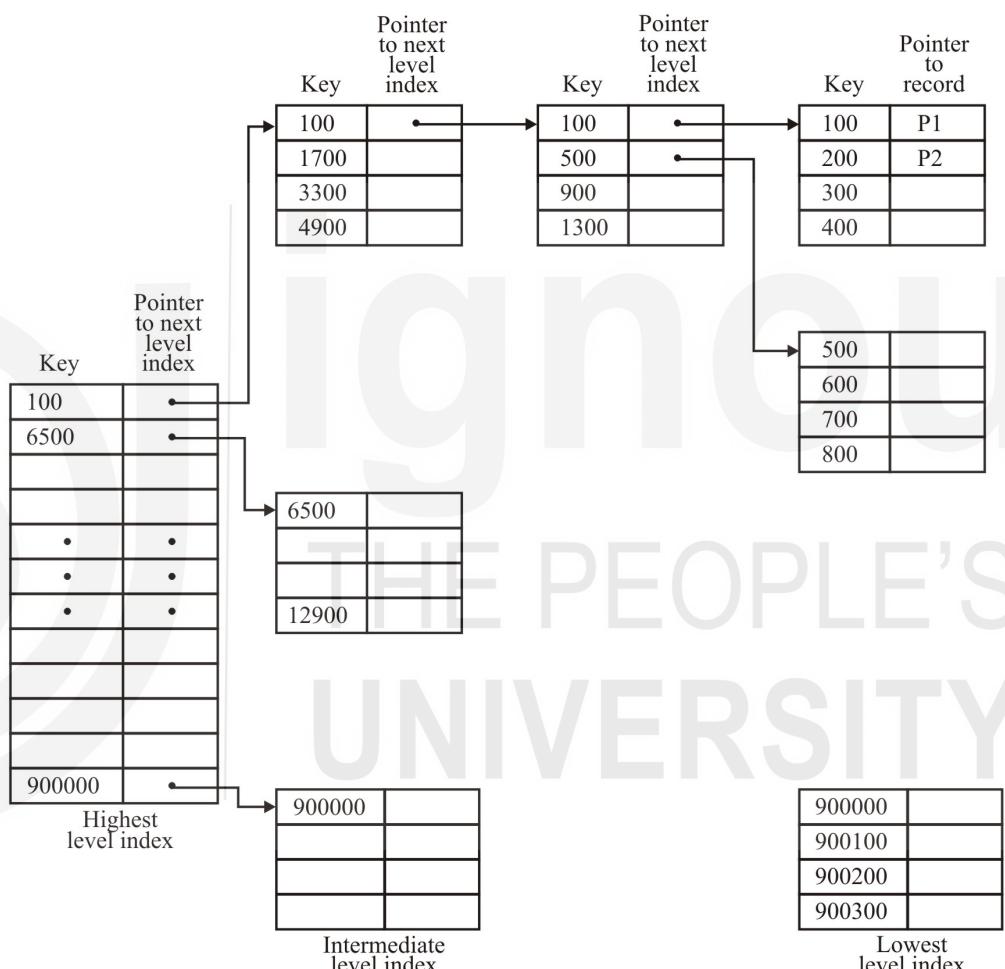


Figure 4.11: Hierarchy of Indexes

Please note the following relating to multi-level indexes:

- The lowest level index, as shown in Figure 4.11, has one entry for each record, though in a different index block. Thus, the lowest level index occupies the maximum space.
 - Thus, maintenance of the multi-level indexes is expensive, as multiple indexes are to be maintained for every insertion and deletion of records.

After discussing so much about the indexes, let us now turn our attention to how an index can be implemented in a database. The indexes are implemented through B-Tree. The following section examines the index implementation in more detail.

Check Your Progress 2

- 1) A file contains 40,000 student records of 200 bytes, each having the 1 KB as the size of a block. What would be the size of the primary index, if the size of the

primary key is 4 byte and the block address is 8 bytes? What would be the average number of block accesses to access a record using this Index?

.....
.....
.....

- 2) What would be the size of secondary index for the student file as stated in question 1 if it has a secondary index on its alternate key of size 8 bytes. What would be the average number of block accesses to access a record using this secondary Index?
-
.....
.....

- 3) Which of the following indexes are dense indexes? Give reasons.
 (i) Primary Index (ii) Clustering Index (iii) Secondary Index on alternative key
 (iv) Secondary index on non-key attributes with unique attribute values
-
.....
.....

4.6 IMPLEMENTING INDEX USING TREE STRUCTURE

Let us discuss the data structure that is used for creating indexes.

Can we use Binary Search Tree (BST) as Indexes?

Let us first reconsider the binary search tree. A BST is a data structure that has a property that all the keys that are to the left of a node are smaller than the key value of the node and all the keys to the right are larger than the key value of the node.

To search a typical key value, you start from the root and move towards left or right depending on the value of the key that is being searched. Since an index is a *<value, address>* pair, thus while using BST, you need to use the value as the key and address field must also be specified in order to locate the records in the file that is stored on the secondary storage devices. The following figure demonstrates the use of BST index for a University where a dense index exists on the enrolment number field.

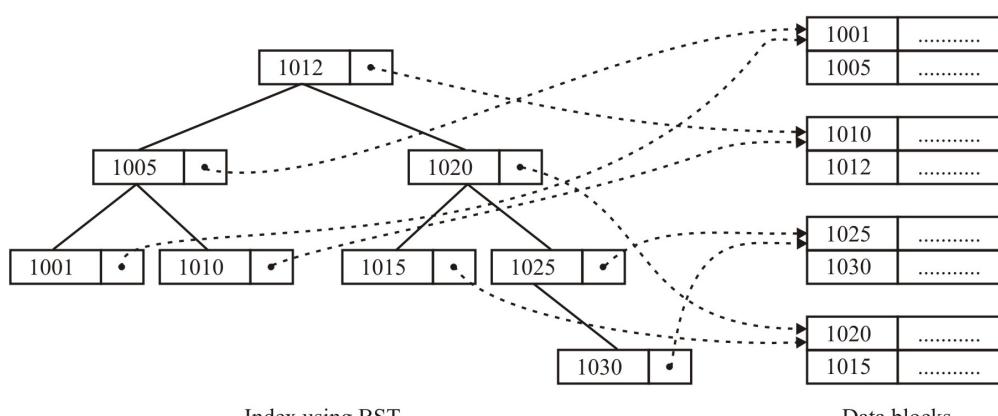


Figure 4.12: The Index structure using Binary Search Tree

Please note in Figure 4.12 that a key value is associated with a pointer to a record. A record consists of the key value and other information fields. Please note that a node on BST stores the *<key value, address>* pair.

Now, let us examine the suitability of BST as a data structure to implement indexes. A BST as a data structure is suitable for an index if the complete index is contained in the primary memory. However, indexes are quite large in nature and require a combination of primary and secondary storage. Therefore, you can use B-Tree data structure to implement the index.

A B-Tree as an index has two advantages:

- It is completely balanced.
- Each node of B-Tree can have a number of keys. An ideal node size of B-Tree would be equal to the block size of the secondary storage device that is being used for storing the index.

The question that needs to be answered here is: what should be the order of B-Tree for an index? The suggested order is from 80-200 depending on various index structures and block size.

B-tree is a data structure, which was proposed by R. Bayer and E. McCreight of Bell Scientific Research Labs in 1970. The B-Tree and its variants are secondary storage structures and have been found to be very useful for implementing indexes. An N order B-tree has:

- A node of B-tree of order N can have children/paths in the range - ceiling of $[N/2]$ to N. However, the root node of the tree can have 2 to N children/paths.
- Each node can have one fewer key than the number of children/paths, but a maximum of $N-1$ keys can be stored in a node.
- The keys are normally arranged in a node in an increasing order.
- If a new key is inserted into a full node of order N (i.e. it already contains $N-1$ keys), then on addition of this new key value, the node would have $N+1$ paths (N keys). This node is split into two nodes and the median key value is moved to the parent of this node. In case the node that is being split is the root node, then it is split into two nodes and a new root node is created by using the median key of the node being split.
- B-tree does not allow any empty sub-tree, therefore, all the leaves of B-tree are at the same level. Therefore, a B-tree is a completely balanced tree.

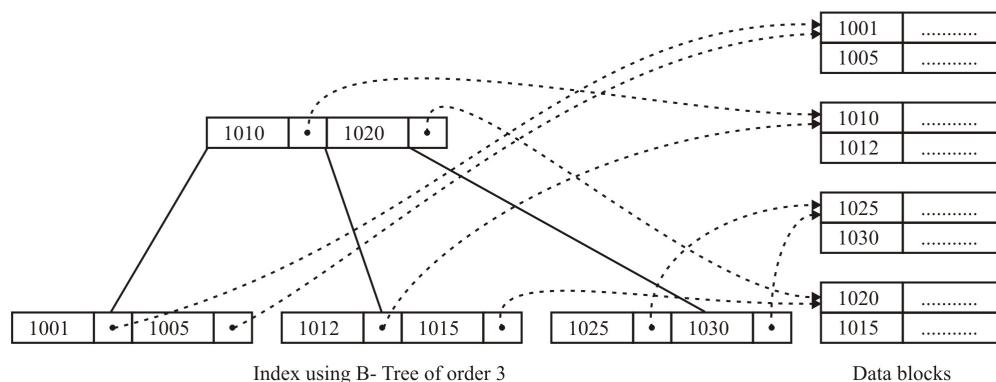


Figure 4.13: A B-Tree as an index

A B-Tree index is shown in Figure 4.13. The B-Tree has a very useful variant called B+Tree, which has all the key values at the leaf level also, in addition to the higher level. For example, the key value 1010 in *Figure 12* will also exist at leaf level. In

addition, these lowest level leaves are linked through pointers. Thus, the B+tree is a very useful structure for index-sequential organisation. You can refer to further readings for more details on these topics.

4.7 MULTI-KEY FILE ORGANISATIONS

Till now we have discussed file organisations having the single access key. But is it possible to have file organisations that allow access of records on more than one key field? This section discusses the two file organisations that allow multiple access paths, with each path having a different key. These are called multi-key file Organisations. These file organisations, in general, are part of a real database management system. Two of the commonest techniques for this Organisation are:

- Multi-list file Organisation
- Inverted file Organisation

Let us discuss these techniques in more detail. But first let us discuss the need for the Multiple access paths.

4.7.1 Multiple Access Paths

In practice, most of the online information systems require the support of multi-key files. For example, consider a banking database application having many kinds of users such as:

- Teller
- Loan officers
- Branch manager
- Account holders

All these users access the bank data however in a different way. Let us assume a sample data format for the Account relation in a bank as:

Account Relation:

Account Number	Account Holder Name	Branch Code	Account type	Balance	Permissible Loan Limit

A teller may access the record above to check the balance at the time of withdrawal. S/he needs to access the account based on branch code and account number. A loan approver may be interested in finding the potential customer by accessing the records in decreasing order of permissible loan limits. A branch manager may need to find the top ten most preferred customers in each category of account, so s/he may access the database in the order of account type and balance. The account holder may be interested in her/his own record. Thus, all these applications are trying to refer to the same data but using different key values. Thus, all the applications as above require the database file to be accessed in different format and order.

Multiple indexes can be used to access a data file through multiple access paths. In such a scheme only one copy of the data is kept, only the number of paths is added with the help of indexes. Let us discuss two important approaches, viz. multi-list file organisation and Inverted file organisation.

4.7.2 Multi-list file Organisation

This file organisation, as the name suggests, consists of multiple lists or indexes. The records in each list are linked from the index value. The linking of records, in general, is done in the sorted sequence of the key attribute to facilitate searching, insertion and deletion operations. The following example explains the multi-list file organisation.

A sample data of employees of an organisation is given in *Figure 4.14*. Assume that the Empid is the key attribute. You can create multiple index lists using this data.

Assumed Record Number	Employee id (Empid)	Employee Name	Job Title	Highest Qualification	Gender (Female F /Male M)	City of posting	Married - M/ Single - S	Salary per month
A	795	Praveen	Engineer	B. Tech.	M	Dehradun	S	16,200/-
B	495	Rohini	Manager	B. Tech.	F	Dehradun	M	19,000/-
C	905	Rishika	Manager	MCA	F	Jaipur	S	17,100/-
D	705	Gaurav	Engineer	B. Tech.	M	Jaipur	M	13,200/-
E	595	Dipti	Manager	MCA	F	Jaipur	S	14,100/-

Figure 4.14: Sample Employee Data

The primary link order (in the order of primary key Empid) would be:

B(495), E(595), D(705), A(795), C(905)

The primary index for this file would be:

≥ 500 but < 700
 ≥ 700 but < 900
 ≥ 900 but < 1100

The index file for the example data as per Figure 4.14 is shown in *Figure 4.15*.

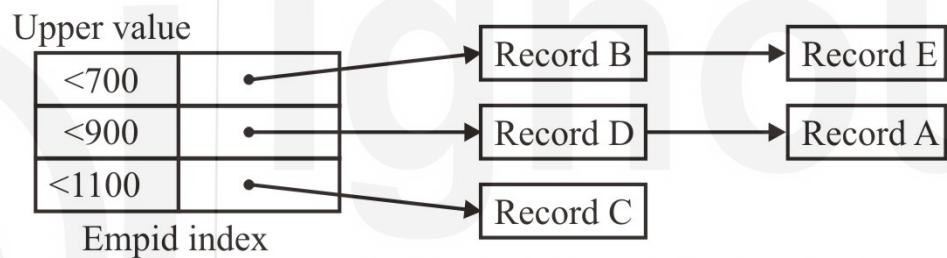


Figure 4.15: Linking together all the records in the same index value.

Please note that in the *Figure 4.15*, those records that fall in the same index value range of Empid are linked together. These lists are smaller than the total range, which will improve search performance.

This file can be supported by many more indexes that will enhance the search performance on various fields, thus, creating a multi-list file organisation. *Figure 4.16* shows various indexes and lists corresponding to those indexes. For simplicity we have just shown the links and not the complete record. Please note that nodes in the original file are assumed to be in the order of Empid's.

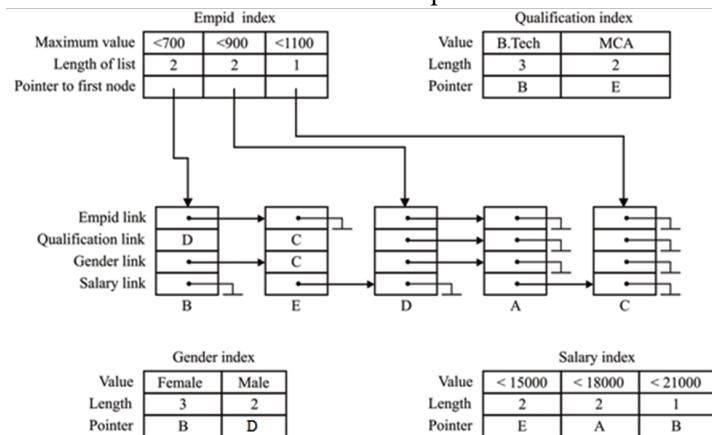


Figure 4.16: Multi-list representation for Figure 4.14

In Figure 4.16, the qualification index can be read as:

B. Tech. Starts at B \rightarrow D \rightarrow A.

MCA Starts at E → C.

Gender Index:

Female: Starts at B, → E, → C.

Male: Starts at B, → D, → A.

An interesting addition that can be done in the indexing scheme of multi-list organisation is that an index entry may contain the length of each sub-list and the index entry should have a pointer to the first record of that list. The length information is useful when the query contains a Boolean expression. For example, if you need to find the list of Female employees who have MCA qualifications, you can find the results in two ways. Either you go to the Gender index and search the Female index list for MCA qualification, or you search the qualification index to find MCA list and in MCA list search for Female candidates. Since the size of MCA list is 2 while the size of Female list is 3 so the preferable search will be through a smaller MCA index list. Thus, the information about the length of the list may help in reducing the search time in the complex queries. Performance of this structure is not good under heavy insertion and deletion of records. However, it is a good structure for searching records in case the appropriate index exists.

4.7.3 Inverted File Organisation

Inverted file organisation is one file organisation where the index structure is most important. In this organisation the basic structure of file records does not matter much. This file organisation is somewhat similar to that of multi-list file organisation with the key difference that in multi-list file organisation index points to a list, whereas in inverted file organisation the index itself contains the list. Thus, maintaining the proper index through proper structures is an important issue in the design of inverted file organisation. Let us show inverted file organisation with the help of data given in *Figure 4.14*.

Let us assume that the inverted file organisation for the data shown contains a dense index. *Figure 4.17* shows how the data can be represented using inverted file organisation.

Empid index		Qualification index		Salary index	
495	B	B.Tech	B, D, A	13200	D
595	E	MCA	E, C	14100	E
705	D	Gender index		16200	A
795	A	Female	B, E, C	17100	C
905	C	Male	D, A	19000	B

Figure 4.17: Some of the indexes for fully inverted file

Please note the following points for the inverted file organisation:

- The index entries are of variable lengths as the number of records with the same key value is changing, thus, maintenance of index is more complex than that of multi-list file organisation.
- The queries that involve Boolean expressions require accesses only for those records that satisfy the query in addition to the block accesses needed for the indices. For example, the query about Female, MCA employees can be solved by the Gender and Qualification index. You just need to take the intersection of record numbers on the two indices. (Please refer to *Figure 4.17*). Thus, any complex query requiring Boolean expression can be handled easily through the help of indices.

4.8 IMPORTANCE OF FILE ORGANISATION IN DATABASES

To implement a database efficiently, there are several design tradeoffs needed. One of the most important ones is the file Organisation. For example, if there were to be an application that required only sequential batch processing, then the use of indexing techniques would be pointless and wasteful.

There are several important consequences of an inappropriate file Organisation being used in a database. The wrong file Organisation will result in:

- much larger processing time for retrieving or modifying the required record.
- undue disk access that could stress the hardware.

Needless to say, there could be many undesirable consequences at the user level, such as making some applications impractical.

Check Your Progress 3

- 1) What is the difference if indexes are implemented using binary search tree or using B-tree?

.....
.....

- 2) What are the advantages of using B+ tree index over B tree index if the supported file organisation is required to access the records sequentially too.

.....
.....
.....

- 3) What is the need of a multi-list organisation? What is the advantage of storing the number of records in an index entry?

.....
.....
.....

4.9 SUMMARY

In this unit, we discussed the physical database design issues in which we had addressed the file Organisation and file access method. The unit also discusses different types of file organization giving their advantages and their disadvantages.

An index is an important component of a database system, as one of the key requirements of DBMS is efficient access to data. This unit explains various types of indexes that may exist in database systems. Some of these are: Primary index, clustering index and secondary index. The secondary index results in better search performance but adds on the task of index updating. This unit also discusses two multi-key file organisations viz. multi-list and inverted file organisations. These are very useful for improving query performance.

4.10 SOLUTIONS / ANSWERS

Check Your Progress 1

1)

Operation	Comments
File Creation	It will be efficient if transaction records are ordered by record key
Record Location	As it follows the sequential approach it is inefficient. On an average, half of the records in the file must be processed to locate a record.
Record Creation	It will require you to browse through all the records to check if such a record already exists or not. Thus, the entire file must be read and written. Efficiency improves if a group of records are created together. This operation could be combined with deletion and modification transactions to achieve greater efficiency.
Record Deletion	The entire file must be read and written. Efficiency improves with greater number of deletions. This operation could be combined with addition and modification transactions to achieve greater efficiency.
Record Modification	Very efficient if the number of records to be modified is high and the records in the transaction file are ordered by the record key.

- 2) Direct-access systems do not search the entire file; rather they move directly to the record, which is to be accessed. To be able to achieve this, several strategies like relative addressing, hashing and indexing can be used.
- 3) It is a technique for physically arranging the records of a file on secondary storage devices. When choosing the file Organisation, you should consider the following factors:
1. Data retrieval speed
 2. Data processing speed
 3. Efficient use of storage space
 4. Protection from failures and data loss
 5. Scalability
 6. Security

Check Your Progress 2

1) Number of accesses without using Primary Index:

Number of records in the file = 40000

Block size = 1024 bytes

Record size = 200 bytes

Number of records per block = integer value of $[1024 / 200] = 05$

Number of disk blocks acquired by the file

$$= [\text{Number of records} / \text{records per block}]$$

$$= [40000/05] = 8000$$

The file is in the order of primary index, so binary search can be employed to access the file. As $2^{13} = 8192$. Thus, about 13 Block accesses would be required to locate the block containing the desired record.

Number of accesses with Primary Index:

Size of an index entry = $4+8 = 12$ bytes
Number of index entries that can be stored per block
= integer value of $[1024 / 12] = 85$
Number of index entries = number of disk Blocks of file = 8000
Number of index blocks = ceiling of $[8000 / 85] = 94$
Number of index block transfers to find the value in index blocks
= ceiling of $\lceil \log_2 94 \rceil = 7$
One block transfer will be required to get the data records using
the index pointer after the required index value has been
located. So total number of block transfers with secondary
index = $7 + 1 = 8$
Thus, the Primary index would save about 5 block transfers for the given case.

2) **Number of accesses without using Secondary Index:**

Number of disk Blocks = 8000 (as computed in Question 1)
The file is in the order of primary index, so search on alternative key
would require on an average half of the Blocks to be searched.
Average number of block transfers (without secondary index) = 4000

Number of accesses with Secondary Index on alternate key (dense index):

Size of an index entry = $8+8 = 16$ bytes
Number of index entries that can be stored per block
= integer value of $[1024 / 16] = 64$
Number of index entries = number of records = 40000
Number of index blocks = ceiling of $[40000 / 64] = 625$
Number of index block transfers to find the value in index blocks
= ceiling of $\lceil \log_2 625 \rceil = 10$
One block transfer will be required to get the data records using
the index pointer after the required index value has been
located. So total number of block transfers with secondary
index = $10 + 1 = 11$

Thus, the Secondary index would save a large number of block transfers.

- 3) (i) Sparse as only one entry per Block of data.
(ii) Dense, as one index entry for each attribute value, sparse if multi-level
index is used.
(iii) Dense, as one index entry for each attribute value
(iv) Dense, as one index entry for each unique attribute

Check Your Progress 3

- 1) In a B+ tree the leaves are linked together to form a sequence set; interior
nodes exist only for the purposes of indexing the sequence set (not to index
into data/records). The insertion and deletion algorithms differ slightly.
- 2) Sequential access to the keys of a B-tree is much slower than sequential
access to the keys of a B+ tree, since the latter have all the keys linked in
sequential order in the leave.
- 3) The multi-list organisation enhances the query answering capability of
databases on multiple key values. The number of records in an index entry
determines the length of the index on a specific attribute value. This enhances
the performance of searching when more than one search term are used, as
you can select the smaller list and then apply second search term on it.

UNIT 5 DATABASE INTEGRITY, FUNCTIONAL DEPENDENCY AND NORMALISATION

Structure	Page Nos.
5.0 Introduction	
5.1 Objectives	
5.2 Database Integrity	
5.2.1 The Keys	
5.2.2 Referential Integrity	
5.2.3 Entity Integrity	
5.3 Redundancy and Associated Problems	
5.4 Functional Dependencies	
5.5 Normalisation Using Functional Dependencies	
5.5.1 The First Normal Form	
5.5.2 The Second Normal Form	
5.5.3 The Third Normal Form	
5.5.4 Boyce Codd Normal Form	
5.6 Desirable Properties of Decomposition	
5.7 Rules of Data Normalisation	
5.7.1 Eliminate Repeating Groups	
5.7.2 Eliminate Redundant Data	
5.7.3 Eliminate Columns Not Dependent on Key	
5.8 Summary	
5.9 Answers/Solutions	

5.0 INTRODUCTION

The first block of this course discusses the database concept, relational algebra, Entity relationship model and integrity constraints in the context of relational database design. One of the key aspects of database design is to create relations without any data redundancy.

This unit first explains the concept of entity and referential integrity. It also explains the anomalies in a relational database system. To remove anomalies is through decomposing the database, you need to decompose relations into smaller relations, which are free of those anomalies. This decomposition may be lossless and dependency preserving. The Unit explains the concept of functional dependency, which is the basis of lossless decomposition of a relation into smaller relations.

The unit deals with the standard form of database relations and discusses the process of decomposing relations into different normal forms up to BCNF. The higher normal forms are discussed in the next unit.

5.1 OBJECTIVES

After going through this unit, you should be able to:

- Explain entity integrity and referential integrity constraints of a relation;
- Explain various anomalies that exist in a database system;
- Define the desirable properties of decomposing a relation;
- Define and use functional dependencies to normalise databases.

5.2 DATABASE INTEGRITY

Database integrity refers to maintaining the consistency of data in the database. Data integrity relates to the correctness of data and often is implemented using constraints on attributes in one or more relations. In this section, we will discuss more about two important integrity constraints of a database: the entity integrity constraint and the referential integrity constraint. To define these two, let us once again define the term Key with respect to a Database Management System.

5.2.1 The Keys

Candidate Key: In a relation R, a candidate key for R is a subset of the set of attributes of R, which have the following two properties:

- | | |
|------------------------|---|
| (1) <i>Uniqueness</i> | No two distinct tuples in R have the same value for the candidate key |
| (2) <i>Irreducible</i> | No proper subset of the candidate key has the uniqueness property. |

Every relation must have at least one candidate key which cannot be reduced further. Duplicate tuples are not allowed in relations. Any candidate key can be a composite key also. For Example, (student-id + course-id) together can form the candidate key of a relation called *Result* (*student-id, course-id, marks*).

Let us summarise the properties of a candidate key.

- A candidate key must be unique and irreducible.
- A candidate may involve one or more than one attribute. A candidate key that involves more than one attribute is said to be composite.

But why are we interested in candidate keys?

Candidate keys are important because they provide the basic **tuple-level identification** mechanism in a relational system. For example, if the enrolment number is the candidate key of a STUDENT relation, then the answer of the query: "Find student details from the STUDENT relation having enrolment number A0123" will output at most one tuple.

Primary Key

The primary key is the candidate key that is chosen by the database designer as the principal means of identifying entities within an entity set. The remaining candidate keys, if any, are called **alternate keys**.

Foreign Keys

Let us first give you the basic definition of foreign key.

Let R2 be a relation, then a foreign key in R2 is a subset of the set of attributes of R2, such that:

1. There exists a relation R1 (R1 and R2 not necessarily distinct) with a candidate key, and
2. For all time, each value of a foreign key in the current state or instance of R2 is identical to the value of Candidate Key in some tuple in the current state of R1.

The definition above seems to be very heavy. Therefore, let us define it in more practical terms with the help of the following example.

Example 1: Assume that in an organisation, an employee may perform different roles in different projects. Say, XYZ is coding in one project and designing in another. Assume that the information is represented by the organisation in three different relations named EMPLOYEE, PROJECT and ROLE. The ROLE relation describes the different roles required in any project.

Assume that the relational schema for the above three relations are:

EMPLOYEE (EMPID, Name, Designation)
PROJECT (PROJID, Proj_Name, Details)
ROLE (ROLEID, Role_description)

In the relations above EMPID, PROJID and ROLEID are unique and not NULL. As you can clearly observe, you can identify the complete instance of the entity set employee through the attribute EMPID. Thus, EMPID is the primary key of the relation EMPLOYEE. Similarly, PROJID and ROLEID are the primary keys for the relations PROJECT and ROLE respectively.

Let ASSIGNMENT is a relationship between entities EMPLOYEE and PROJECT and ROLE, describing which employee is working on which project and what the role of the employee is in the respective project. Figure 5.1 shows part of E-R diagram for these entities and relationships (for simplicity, no attribute has been shown).

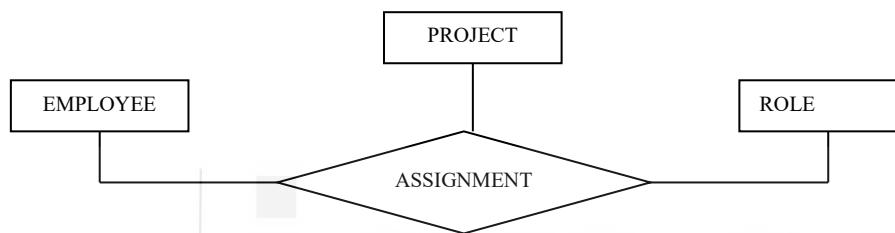


Figure 5.1: E-R diagram for employee role in development

Let us consider sample relation instances as:

EMPLOYEE

EMPID	Name	Designation
101	XYZ	Analyst
102	ABC	Receptionist
103	ARVIND	Manager

PROJECT

PROJID	Proj_name	Details
TCS	Traffic Control System	For traffic shaping.
LG	Load Generator	To simulate load for input in TCS.
B++1	B++ TREE	ISS/R turbo sys

ROLE

ROLEID	Role description
1000	Design
2000	Coding
3000	Marketing

ASSIGNMENT

EMPID	PROJID	ROLEID
101	TCS	1000
101	LG	2000
102	B++1	3000

Figure 5.2: An example relation

You can define the relational scheme for the relation ASSIGNMENT as follows:

ASSIGNMENT (EMPID, PROJID, ROLEID)

Please note now that in the relation ASSIGNMENT (as per the definition of foreign key to be taken as R2) EMPID is the foreign key in ASSIGNMENT relation; it references the relation EMPLOYEE (as per the definition to be taken as R1) where EMPID is the primary key. Similarly, PROJID and ROLEID in the relation ASSIGNMENT are **foreign keys** referencing the relation PROJECT and ROLE respectively.

Now after defining the concept of foreign key, we can proceed to discuss the actual integrity constraints namely Referential Integrity and Entity Integrity.

5.2.2 Referential Integrity

It can be simply defined as:

The database must not contain any unmatched foreign key values.

The term “unmatched foreign key value” means a foreign key value for which there does not exist a **matching value** of the relevant candidate key in the relevant target (referenced) relation. For example, any value existing in the EMPID attribute in ASSIGNMENT relation must exist in the EMPLOYEE relation. That is, the only EMPIDs that can exist in the EMPLOYEE relation are 101, 102 and 103 for the present state/ instance of the database given in *Figure 5.2*. If we want to add a tuple with EMPID value 104 in the ASSIGNMENT relation, it will cause violation of referential integrity constraint. Logically it is obvious, after all the employee 104 does not exist, so how can s/he be assigned any work.

Database modifications can cause violations of referential integrity. We list here the referential action that you may specify for each type of database modification to preserve the referential-integrity constraint:

Delete

During the deletion of a tuple two cases can occur:

Deletion of tuple in relation having the foreign key: In such a case simply delete the desired tuple. For example, in ASSIGNMENT relation you can easily delete the first tuple.

Deletion of the target of a foreign key reference: For example, an attempt to delete an employee tuple in EMPLOYEE relation whose EMPID is 101. This employee appears not only in the EMPLOYEE but also in the ASSIGNMENT relation. Can this tuple be deleted? If you delete the tuple in EMPLOYEE relation then two unmatched tuples are left in the ASSIGNMENT relation, thus causing violation of referential integrity constraint. Thus, the following two choices exist for such deletion:

RESTRICT – The delete operation is “restricted” to only the case where there are no matching tuples in the referencing relation. For example, you can delete the EMPLOYEE record of EMPID 103 as no matching tuple in ASSIGNMENT but not the record of EMPID 101.

CASCADE – The delete operation “cascades” to delete those matching tuples also. For example, if the delete mode is CASCADE, then deleting an employee data having EMPID as 101 from EMPLOYEE relation will also cause deletion of 2 more tuples from ASSIGNMENT relation.

Insert

The insertion of a tuple in the target of reference does not cause any violation. However, insertion of a tuple in the relation in which we have the foreign key, for example, in ASSIGNMENT relation it needs to be ensured that all matching target candidate key exist; otherwise, the insert operation can be rejected. For example, one of the possible ASSIGNMENT insert operations would be (103, LG, 3000).

Modify

Modifying or updating operations changes the existing values. If these operations change the value that is the foreign key also, the only check required is the same as that of the Insert operation.

What should happen to an attempt to update a candidate key that is the target of a foreign key reference? For example, an attempt to update the PROJID “LG” for which there exists at least one matching ASSIGNMENT tuple? In general, there are the same possibilities as for DELETE operation:

RESTRICT: The update operation is “restricted” to the case where there are no matching ASSIGNMENT tuples. (It is rejected otherwise).

CASCADE – The update operation “cascades” to update the foreign key in those matching ASSIGNMENT tuples also.

5.2.3 Entity Integrity

Before describing the second type of integrity constraint, viz., Entity Integrity, you should be familiar with the concept of **NULL**.

Basically, NULL is intended as a basis for dealing with the problem of missing information. This kind of situation is frequently encountered in the real world. For example, historical records sometimes have entries such as “Date of birth unknown”. Hence it is necessary to have some way of dealing with such situations in database systems. Codd proposed an approach to this issue that makes use of special markers called NULL to represent such missing information.

A given attribute in the relation might or might not be allowed to contain NULL. But can the Primary key or any of its components (in case primary key is a composite key) contain a NULL? To answer this question an **Entity Integrity Rule** states: **No component of the primary key of a relation is allowed to accept NULL.** In other words, the definition of every attribute involved in the primary key of any basic relation must explicitly or implicitly include the specifications of **NULL NOT ALLOWED**.

Foreign Keys and NULL

Let us consider the relations:

DEPT		
DEPT ID	DNAME	BUDGET
D1	Marketing	10M
D2	Development	12M
D3	Research	5M

EMP			
EMP ID	ENAME	DEPT ID	SALARY
E1	Rohan	D1	40K
E2	Aparna	D1	42K
E3	Ankit	D2	30K
E4	Sangeeta		35K

Suppose that Sangeeta is not assigned any Department. In the EMP tuple corresponding to Sangeeta, therefore, there is no genuine department number that can serve as the appropriate value for the DEPTID foreign key. Thus, one cannot determine DNAME and BUDGET for Sangeeta’s department as those values are NULL. This may be a real situation where the person has newly joined and is undergoing training and will be allocated to a department only on completion of the training. Thus, NULL in foreign key values may not be a logical error.

So, the foreign key definition may be redefined to include NULL as an acceptable value in the foreign key for which there is no need to find a matching tuple.

Check Your Progress 1

Consider the following relations:

S		
SNO	SNAME	CITY
S1	Smita	Delhi
S2	Jim	Pune
S3	Ballav	Pune

P			
PNO	PNAME	COLOUR	CITY
P1	Nut	Red	Delhi
P2	Bolt	Blue	Pune
P3	Part1	White	Mumbai

S4	Seema	Delhi
S5	Salim	Agra

P4	Part2	Blue	Delhi
P5	Camera	Brown	Pune
P6	Part3	Grey	Delhi

J		
JNO	JNAME	CITY
J1	Sorter	Pune
J2	Display	Bombay
J3	OCR	Agra
J4	Console	Agra
J5	RAID	Delhi
J6	EDP	Udaipur
J7	Tape	Delhi

SPJ			
SNO	PNO	JNO	QUANTITY
S1	P1	J1	200
S1	P1	J4	700
S2	P3	J2	400
S2	P2	J7	200
S2	P3	J3	500
S3	P3	J5	400
S4	P4	J3	900

- 1) For each of the relations, as given above, list the candidate keys. Also, identify the Primary key to each of the relations.

.....
.....
.....
.....

- 2) List the entity integrity constraints, which can be found in relations S, P, J, SPJ?
List the domain constraints, if any.

.....
.....
.....
.....

- 3) Which of the relations S, P, J, SPJ has referential constraints? List those constraints.

.....
.....
.....
.....

- 4) For the referential constraints as identified in question 3, suggest suitable referential actions.

.....
.....
.....
.....

5.3 REDUNDANCY AND ASSOCIATED PROBLEMS

Let us consider the following relation STUDENT.

Enrolment Number (StEnroNo)	Student Name (StName)	Student Address (StAddress)	Course Number (CoNo)	Course Name (CoName)	Instructor (CoInstructor)	Office number of the Instructor (InOffice)
050112345	Rohan	D-27, Main Road, Ranchi	MCS-201	Problem Solution	Nayan Kumar	102
050112345	Rohan	D-27, Main Road, Ranchi	MCS-202	Computer Organisation	Anurag Sharma	105
050112345	Rohan	D-27, Main Road, Ranchi	MCS-203	OS	Preeti Anand	103
050111341	Aparna	B-III, Gurgaon	MCS-203	OS	Preeti Anand	103

Figure 5.3: A state of STUDENT relation

The above relation satisfies the properties of a relation and contains a single value in each cell. Conceptually it is convenient to have all the information in one relation, as a single query to the database may produce complete information about a person. Does the student relation, as given in Figure 5.3 has any undesirable characteristics?

You may observe that Figure 5.3 contains duplicate information in several attributes. For example, the student, whose enrolment number is 050112345 has a name - Rohan and the student stays at an address “D-27, Main Road, Ranchi”. This information is repetitive in the first three attributes of tuples 1, 2 and 3 (shown in Figure 5.3 in red colour). Similarly, the information that course name for number MCS-203 is OS and it is taught by “Preeti Anand”, whose office number is 103 (shown in Figure 5.3 in purple colour). You can observe that even this information is repetitive in tuple 3 and tuple 4. Thus, the relation of Figure 5.3 has the undesirable **Data Redundancy**.

In addition, when a new student takes admission and enrolls for the course MCS-203, then the entire information about the MCS-203 will be added to the relation. Such repetitive information not only increases the size of the database, but also results in several problems, which are discussed below.

1. *Update Anomaly*: Consider the data redundancy of student name and address, as shown in Figure 5.3. Assume that the student Rohan shifts from Ranchi to New Delhi at a new address “F-102, Maidan Garhi, New Delhi”. This will require updating the address of Rohan in all the three tuples of the Student relation. All the three tuples must be updated consistently. If this does not happen, then the address of Rohan will be inconsistent. This inconsistency is the result of update operation on redundant data. Thus, this anomaly is termed an update anomaly, which causes **data inconsistency**.

2. *Insertion Anomaly*: The note that the primary key of the Student relation, as given in Figure 5.3, is composite key consisting of enrolment number and Course Number. Any new tuple to be inserted in the relation must have a value for both the attributes of the primary key, as entity integrity constraint requires that a key may not be totally or partially NULL. However, in the given relation if you want to insert the course number and course name of a new course in the database, it would not be possible until a student enrolls in that course. Similarly, information about a new student cannot be inserted in the database until the student enrolls in a course. These problems are called insertion anomalies.

3. *Deletion Anomalies*: In some instances, useful information may be lost when a tuple is deleted. For example, if you delete the tuple corresponding to student 050111341 enrolled for MCS-203, you will lose relevant information about the

student viz. enrolment number, name and address of this student. Similarly, deletion of tuple having Student Name “Rohan” and Course Number ‘MCS-202’ will result in loss of information that MCS-202 is named “Computer Organisation” having an instructor “Anurag Sharma”, whose office number is 105. This is called deletion anomaly.

The anomalies arise primarily because the relation STUDENT has information about students as well as courses. One solution to the problems is to decompose the relation into two or more smaller relations, so that a relation contains information about one thing. But what should be the basis of this decomposition? To answer the questions let us try to articulate dependence of data within a relation with the help of the following *Figure 5.4*:

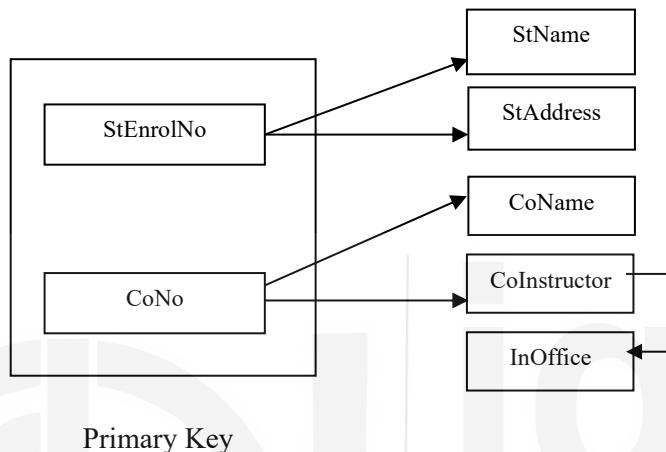


Figure 5.4: The dependencies of relation in Figure 5.3

In *Figure 5.4*, an arrow shows dependence of data attributes. For example, you may notice that two arrows (links) are originating from an attribute student enrolment number (*StEnrolNo*). One of these links is to student name attribute (*StName*) and the second link to student address (*StAddress*). The link between attribute *StEnrolNo* and student name attribute (*StName*) denotes that enrolment number attribute uniquely determines the student’s name. For example, in Figure 5.3 the enrolment number 050112345 uniquely determines that name of the student is Rohan. Likewise, you may determine the dependence among different attributes. You may notice that the dependence of data can exist even among the attributes which are not the part of the primary key, such as, a dependence from course instructor (*CoInstructor*) attribute to instructor office number (*InOffice*) attribute. The dependence among the attributes forms the basis of decomposition of a relation into two or more relations, this is called the normalisation. The objective of this decomposition is to reduce the three anomalies in the decomposed relations. However, normalisation should not result in loss of information, which means that you should be able to obtain the original relation’s information by taking JOIN of the relations obtained after decomposition.

A relation that needs to be normalised may have a very large number of attributes. In such relations, it is almost impossible for a person to conceptualise all the information and suggest a suitable decomposition to overcome the problems. Such relations need an algorithmic approach for finding if there are problems in a proposed database design and how to eliminate them if they exist. The discussions of these algorithms are beyond the scope of this Unit, but we will first introduce you to the basic concept that supports the process of Normalisation of large databases. So let us first define the concept of functional dependence in the subsequent section and follow it up with the concepts of normalisation.

5.4 FUNCTIONAL DEPENDENCIES

A database is a collection of related information and it is therefore inevitable that some items of information in the database would depend on some other items of information. The information is either single-valued or multi-valued. The enrolment number of a student and his/her date of birth are single-valued information; qualifications of a person or subjects that an instructor teaches are multi-valued facts. In this section, we will deal with single-valued facts, which forms the basis of the concept of functional dependency. Let us define this concept logically.

Functional Dependency (FD)

Let us consider a single universal relation schema “A”. A functional dependency denoted by $X \rightarrow Y$, between two sets of attributes X and Y that are subset of universal relation “A” specifies a constraint on the possible tuples that can form a relational state of “A”. Consider any two tuples of a relation A, say t_1 and t_2 , a FD is said to exist between two sets of attributes X to Y, if the following holds in A:

If $t_1(X) = t_2(X)$, then $t_1(Y) = t_2(Y)$ must be true.

It means that, if tuple t_1 and tuple t_2 have same values for attributes X, then to hold $X \rightarrow Y$ on “A”, t_1 and t_2 must have same values for attributes Y also.

Thus, FD $X \rightarrow Y$ means that the values of the Y component of a tuple in “A” depend on or is determined by the values of X component. In other words, the value of the Y component is uniquely determined by the value of the X component. This is functional dependency from X to Y (**but not Y to X**) that is, Y is functionally dependent on X.

The relation schema “A” determines the function dependency of Y on X ($X \rightarrow Y$) when and only when:

- 1) if tuples in “A”, which agree on their X value, then
- 2) they **must** agree on their Y values too.

Please note that if $X \rightarrow Y$ in “A”, does not mean $Y \rightarrow X$ in “A”.

Please note that functional dependency (FD) does not imply a one-to-one relationship between X and Y.

For example, the functional dependencies of Figure 5.4 are:

$$\begin{array}{lcl} \text{StEnrolNo} & \rightarrow & \text{StName, StAddress} \\ \text{CoNo} & \rightarrow & \text{CoName, CoInstructor} \\ \text{CoInstructor} & \rightarrow & \text{InOffice} \end{array}$$

These functional dependencies imply that there can be only one student name for each **StEnrolNo**, only one address for each student and only one course name for each **CoNo**. Please note, given this set of FDs, it is possible that two or more students, who have different enrolment numbers may have the same name. In addition, two or more students can reside at the same address. However, two different students cannot have the same enrolment number.

Similarly, consider **CoNo → CoInstructor**, the dependency implies that no subject can have more than one instructor (perhaps this is not a very realistic assumption). Functional dependencies therefore place constraints on what information the database may store. In addition, in the example above, you may be wondering if the following FDs hold:

$$\text{StName} \rightarrow \text{StEnrolNo} \quad (1)$$

Certainly, there is nothing in the given instance of the database relation presented that contradicts the functional dependencies as above. However, whether these FDs hold or not would depend on whether the university or college, whose database you are considering, allows two different students to have the same name and two different courses to have the same course names. If it was the enterprise policy to have unique course names, then (2) holds. If two students have exactly the same name, then (1) does not hold.

Let us use an E-R diagram to show various FDs (Please refer to *Figure 5.5*). A simple example of functional dependency in this ERD is when X is a primary key of an entity (e.g., enrolment number) and Y is some single-valued property or attribute of the entity (e.g., student name). $X \rightarrow Y$ then must always hold. (Why?)

Functional dependencies also arise in relationships. Let C and D be the primary keys of two strong entity sets participating in a relationship. If the relationship is one-to-one, both the FDs $C \rightarrow D$ and $D \rightarrow C$ will hold. If the relationship is many-to-one (C on many side), an FD $C \rightarrow D$ will hold but the FD $D \rightarrow C$ will NOT hold. In case, a relationship cardinality is many-to-many, then the key attributes of the participating entities would not have any functional dependency between them.

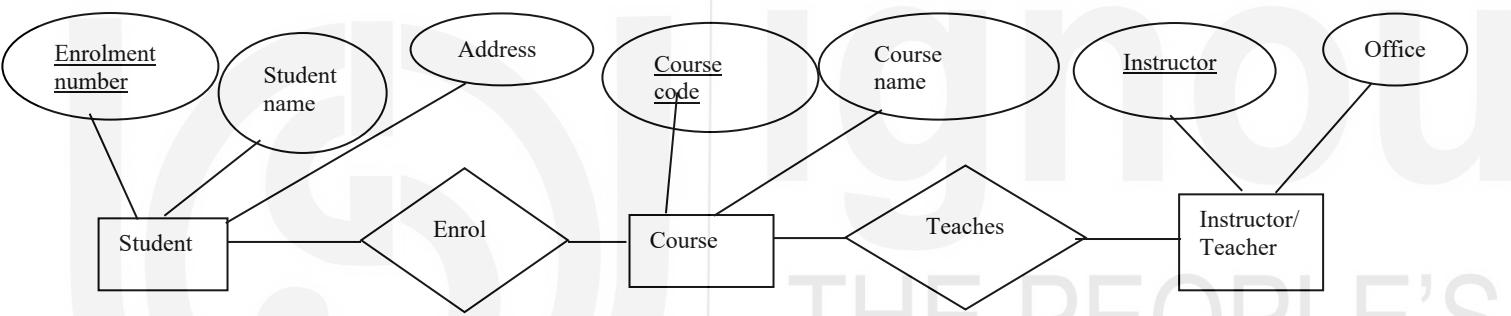


Figure 5.5: ERD of Entities – Student, Course and Teacher

ER diagram Figure 5.5 can be converted to the following relations:

Relations of Entities

STUDENT (StEnrolNo, StName, StAddress)

FDs: StEnrolNo \rightarrow StName, StAddress

COURSE (CoNo, CoName)

FD: CoNo \rightarrow CoName

INSTRUCTOR (CoInstructor, InOffice) //CoInstructor is instructor Id

FD: inId \rightarrow InOffice

Relations of Relationships

ENROL (StEnrolNo, CoCode) (assuming many-to-many cardinality)

FD: StEnrolNo, CoNo \rightarrow NULL

Assuming that one course can be taught by only one teacher, but one teacher can teach many courses, you need to redesign COURSE relation as:

COURSE (CoNo, CoName, CoInstructor)

FDs: CoNo \rightarrow CoName, CoInstructor

Identification of FDs:

Identification of FDs, in general, is not trivial. You need to study the domain of attributes and relationships among these attributes. You cannot identify FDs by using a set of rules. The following example explains this.

Consider the following relation:

STUDENT-COURSE (enrolno, sname, cname, classlocation, hours)

The following functional dependencies may exist on this relation (you should identify the FDs primarily from constraints, there is no thumb rule to do so otherwise):

- **enrolno → sname** (the enrolment number of a student uniquely determines the student names alternatively; you can say that sname is functionally determined/dependent on enrolment number).
- **classcode → cname, classlocation,** (the value of a class code uniquely determines the class name and class location).
- **enrolno, classcode → Hours** (a combination of enrolment number and class code values uniquely determines the number of hours and students' study in the class per week (Hours)).

The semantic property of functional dependency explains how the attributes in “A” are related to one another. A FD in “A” must be used to specify constraints on its attributes that must hold at all times.

For example, a FD (State, City, Place) → Pin_code should hold for any address in India. It is also possible that certain functional dependencies may cease to exist in the real world if the relationship changes, for example, the FD Pin_code → Area_code used to exist as a relationship between postal codes and telephone number area codes in India, however, with the proliferation of mobile telephone, the FD is no longer true.

The set of FDs over a relation can be optimised to obtain a minimal set of FDs called the canonical cover. However, these topics are beyond the scope of this course and can be studied by consulting further readings.

5.5 NORMALISATION USING FUNCTIONAL DEPENDENCIES

Codd in the year 1972 presented three normal forms (1NF, 2NF, and 3NF). These were based on functional dependencies among the attributes of a relation. Later Boyce and Codd proposed another normal form called the Boyce-Codd normal form (BCNF). The fourth and fifth normal forms are based on multi-valued dependency and join dependencies and were proposed later. In this section we will cover normal forms till BCNF only. Fourth and fifth normal forms are discussed in the next unit. For all practical purposes, 3NF or the BCNF are quite adequate since they remove the anomalies discussed for most common situations. It should be clearly understood that there is no obligation to normalise relations to the highest possible level. Performance should be taken into account and sometimes an organisation may take a decision not to normalise, say, beyond third normal form. But it should be noted that such designs should be careful enough to take care of anomalies that would result because of the decision above.

Intuitively, the second and third normal forms are designed to result in relations such that each relation contains information about only one thing (either an entity or a relationship). A sound E-R model of the database would ensure that all relations either provide facts about an entity or about a relationship resulting in the relations that are obtained being in 2NF or 3NF.

The normalisation of a relation till BCNF is established using the FDs. The normalisation process depends on the assumptions that:

- 1) a set of functional dependencies is given for each relation, and
- 2) Each relation has a designated primary key.

The normalisation process proceeds in a top-down fashion by evaluating each relation against the criteria for normal forms and decomposing relations as found necessary during analysis. Therefore, normalisation upto BCNF is looked upon as a process of analysing the given relation schemas based on their condition (FDs and Primary Keys) to achieve the desirable properties:

- firstly, minimising redundancy, and
- secondly minimising the insertion, deletion and update anomalies.

Thus, the normalisation provides the database designer with:

- 1) a formal framework for analysing relation schemas.
- 2) a series of normal form tests that can be normalised to any desired degree.

Decomposition through normalisation should include any or both of the following properties.

- 1) the lossless join and non-additive join property, and
- 2) the dependency preservation property.

Do you always normalise the database to the highest normal form? Due to performance reasons, database designers sometimes leave relations in lower normalisation forms. This is called denormalisation.

Let us now define the normal forms in more detail.

5.5.1 The First Normal Form (1NF)

Let us first define 1NF:

Definition: A relation (table) R is in 1NF if every attribute of R takes atomic values. In other words, the following conditions hold in R:

1. There are no duplicate rows or tuples in the relation.
2. Each data value stored in the relation is single-valued.
3. Entries in a column (attribute) are of the same kind (type).

Please note that in a 1NF relation, the order of the tuples (rows) and attributes (columns) does not matter.

The first requirement above means that the relation **must have a key**. The key may be single attribute or composite key. It may even, possibly, contain all the columns.

The first normal form defines only the basic structure of the relation and does not resolve the anomalies discussed in Section 5.3.

The relation STUDENT (StEnrolNo, StName, StAddress, CoNo, CoName, CoInstructor, InOffice) of *Figure 5.3* is in 1NF. The primary key of the relation is a composite key of attributes StEnrolNo and CoNo.

5.5.2 The Second Normal Form (2NF)

The relation of *Figure 5.3* is in 1NF, yet it suffers from all the anomalies. Therefore, a second normal form may be defined to overcome the anomalies.

Definition: A relation is in Second Normal Form (2NF) if it fulfills the following criteria (assuming the relation has only one candidate key, which is selected as the primary key of the relation):

- (i) The relation fulfills the criteria of the First Normal Form (1NF), and
- (ii) All those attributes, which are not part of any primary key, are fully functionally dependent on the primary keys.

Key features of 2NF:

- The partial dependency will exist, only if the primary key is composite. Therefore, if the primary key of a relation consists of a single attribute, then the given 1NF relation would be in 2NF also.
- This normal form decomposes a relation so that relations store information about one thing.

Please refer to *Figure 5.4*, which illustrates the FDs in the relation STUDENT (StEnrolNo, StName, StAddress, CoNo, CoName, CoInstructor, InOffice). The FDs, as shown in *Figure 5.4* are:

$$\begin{array}{lcl} \text{StEnrolNo} & \rightarrow & \text{StName, StAddress} \\ \text{CoNo} & \rightarrow & \text{CoName, CoInstructor} \\ \text{CoInstructor} & \rightarrow & \text{InOffice} \end{array} \quad \begin{array}{l} (1) \\ (2) \\ (3) \end{array}$$

Since, from the above FDs, you can create a FD:

$$\text{StEnrolNo, CoNo} \rightarrow \text{StName, StAddress, CoName, CoInstructor, InOffice}$$

Therefore, the attributes StEnrolNo and CoNo together form the composite key to the relation STUDENT. The relation has only one candidate key, therefore, all the attributes, other than StEnrolNo and CoNo, of the relation are not part of a candidate key. These attributes are also called non-key attributes. The relation STUDENT is in 1NF, but is it in 2NF? No, the FDs at (1) and (2) are clearly violating the fully functionally dependent criteria of 2NF (*Figure 5.4*). Therefore, the relation suffers from the anomalies as shown in Figure 5.3. Next, how will you decompose the STUDENT relation to 2NF relations? You may use FDs of (1), (2) and (3) to do so. FD (1) can be used to create a 2NF relation consisting of these three attributes from STUDENT relation. This part new relation is:

$$\text{STUDENT1 } (\underline{\text{StEnrolNo}}, \text{StName}, \text{StAddress})$$

Similarly, you can use FD (2) to decompose the STUDENT relation further, but what about the attribute ‘InOffice’? You find in FD (2) that Course code (CoNo) attribute uniquely determines the name of instructor (refer to FD 2(a)). Also, the FD (3) means that the name of the instructor uniquely determines the office number. This can be written as:

$$\begin{array}{lcl} \text{CoNo} & \rightarrow & \text{CoInstructor} \\ \text{CoInstructor} & \rightarrow & \text{InOffice} \end{array} \quad \begin{array}{l} (2 \text{ (a)) (without CoName)} \\ (3) \end{array}$$

The above two FDs imply a transitive dependency:

$$\text{CoNo} \rightarrow \text{InOffice} \quad (\text{This is transitive dependency})$$

The revised FD (2) is:

$$\text{CoNo} \rightarrow \text{CoName, CoInstructor, InOffice} \quad (4)$$

Use this FD to create another 2NF relation:

$$\text{COU_INST } (\underline{\text{CoNo}}, \text{CoName}, \text{CoInstructor}, \text{InOffice})$$

Now, you have the following two 2NF relations, which are obtained by decomposing the STUDENT relation:

$$\begin{array}{l} \text{STUDENT1 } (\underline{\text{StEnrolNo}}, \text{StName}, \text{StAddress}) \\ \text{COU_INST } (\underline{\text{CoNo}}, \text{CoName}, \text{CoInstructor}, \text{InOffice}) \end{array}$$

Are these two 2NF relations sufficient to represent the relations STUDENT? No, as there is no common attribute between the relations. Therefore, they cannot be joined together to get the data of STUDENT relation. You must have a relation that joins the two decomposed relations. This relation would have the primary key attributes of the STUDENT relation and any other attribute that is fully

functionally dependent on this primary key. However, there is no attribute in STUDENT relation that is fully dependent on the composite primary key. Thus, you will create a third relation using the composite primary key of the STUDENT relation, as shown below:

COURSE_STUDENT (StEnrolNo, CoNo)

Please note that the COURSE_STUDENT relation can be joined with STUDENT1 relation on StEnrolNo attribute and with COU_INST relation on CoNo attribute.

The relation STUDENT in 2NF form would be:

STUDENT1 (<u>StEnrolNo</u> , StName, StAddress)	2NF(a)
COU_INST (<u>CoNo</u> , CoName, CoInstructor, InOffice)	2NF(b)
COURSE_STUDENT (<u>StEnrolNo</u> , <u>CoNo</u>)	2NF(c)

5.5.3 The Third Normal Form (3NF)

Although, transforming a relation that is not in 2NF into a number of relations that are in 2NF removes many of the anomalies, it does not necessarily remove all anomalies. Thus, further Normalisation is sometimes needed to ensure further removal of anomalies. These anomalies arise because a 2NF relation may have attributes that are not directly related to the primary key of the relation.

Definition: A relation is in third normal form if it is in 2NF and every non-key attribute of the relation is non-transitively dependent on the primary key of the relation.

But what is **non-transitive** dependence?

Let A, B and C be three attributes of a relation R such that $A \rightarrow B$ and $B \rightarrow C$. From these FDs, we may derive $A \rightarrow C$. This dependence $A \rightarrow C$ is transitive.

Now, let us reconsider the relation 2NF (b)

COU_INST (CoNo, CoName, Instruction, InOffice)

Assume that CoName is not unique and therefore CoNo is the only candidate key. The following functional dependencies exist:

CoNo	\rightarrow	CoInstructor	(2 (a))
CoInstructor	\rightarrow	InOffice	(3)
CoNo	\rightarrow	InOffice	(This is transitive dependency)

You had derived $CoNo \rightarrow InOffice$ from the functional dependencies 2(a) and (3) for decomposition to 2NF. The relation is, however, not in 3NF since the attribute 'InOffice' is not directly dependent on attribute 'CoNo' but is transitively dependent on it and should, therefore, be decomposed as it has all the anomalies. The primary difficulty in the relation above is that an instructor might be responsible for several subjects, requiring one tuple for each course. Therefore, his/her office number will be repeated in each tuple. This leads to all the problems such as update, insertion, and deletion anomalies. To overcome these difficulties, you need to decompose the relation 2NF(b) into the following two relations:

COURSE (CoNo, CoName, CoInstructor) INST (CoInstructor, InOffice)

Please note these two relations and 2NF (a) and 2NF (c) are already in 3NF. Thus, the relation STUDENT in 3 NF would be:

STUDENT1 (StEnrolNo, StName, StAddress)
COURSE (CoNo, CoName, CoInstructor)
INST (CoInstructor, InOffice)
COURSE_STUDENT (StEnrolNo, CoNo)

The 3NF is usually quite adequate for most relational database designs. There are, however, some situations where a relation may be in 3 NF but have anomalies. For example, consider a relation STUDENT5 (StEnrolNo, StName, CoNo, CoName). Assume it has the following set of FDs:

$$\begin{array}{l} \text{StEnrolNo} \rightarrow \text{StName} \\ \text{StName} \rightarrow \text{StEnrolNo} \\ \text{CoNo} \rightarrow \text{CoName} \\ \text{CoName} \rightarrow \text{CoNo} \end{array}$$

Is the relation STUDENT5 is in 3NF? The FDs of this relation can be written as:

$$\begin{array}{l} \text{StEnrolNo, CoNO} \rightarrow \text{StName, CoName} \\ \text{StEnrolNo, CoName} \rightarrow \text{StName, CoNO} \\ \text{StName, CoNO} \rightarrow \text{StEnrolNo, CoName} \\ \text{StName, CoName} \rightarrow \text{StEnrolNo, CoNO} \end{array}$$

Therefore, the relation STUDENT5 has the following candidate keys.

(StEnrolNo, CoNo)
(StEnrolNo, CoName)
(StName, CoNo)
(StName, CoName)

Figure 5.6 shows the functional dependency diagram for this relation.

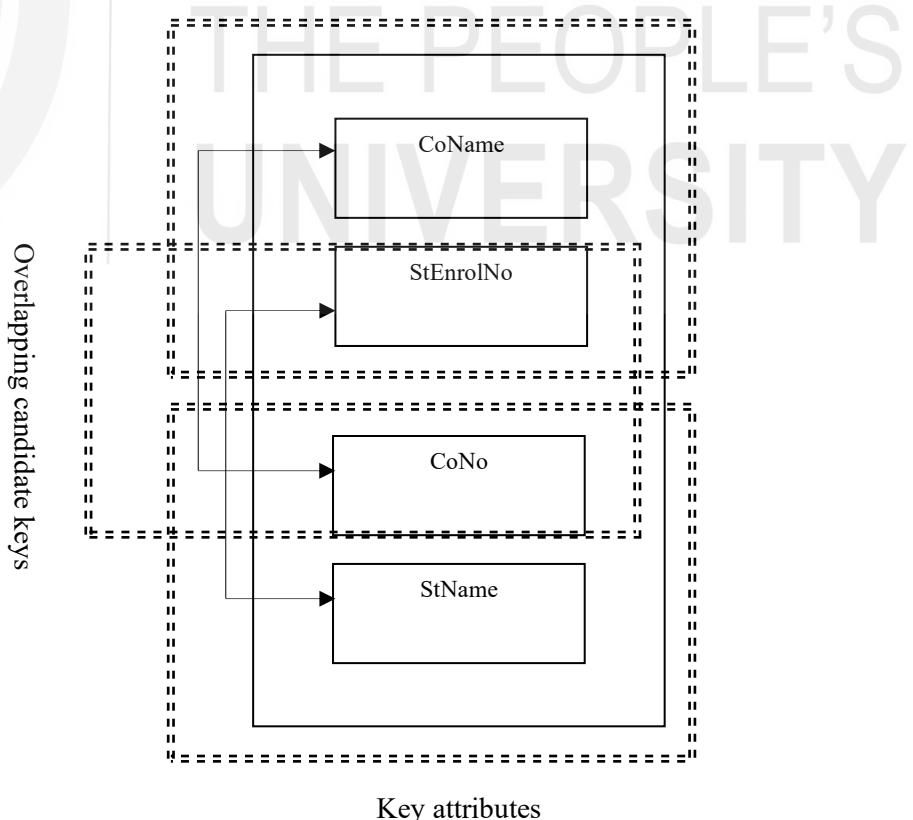


Figure 5.6: Functional Diagram for STUDENT5 relation

You may observe that all the attributes of STUDENT5 relations are prime attributes as they are part of some candidate key. You may also observe that the relation has overlapping candidate keys. Therefore, no non-key attribute exists in the relation. Hence, the relation follows the criteria of 2NF and 3NF and is in 3NF. However, this relation still suffers from the three anomalies (please make an instance for the relation), as the attributes in the non-overlapping part of the candidate key can determine each other. Therefore, the relation is to be normalised into a stronger normal form called BCNF.

5.5.4 Boyce-Codd Normal Form (BCNF)

As stated earlier, the relation STUDENT5 (StEnrolNo, StName, CoNo, CoName) has the following candidate keys:

$$\begin{array}{ll} (\text{StEnrolNo}, \text{CoNo}) ; & (\text{StEnrolNo}, \text{CoName}) \\ (\text{StName}, \text{CoNo}) ; & (\text{StName}, \text{CoName}) \end{array}$$

Since the relation has no non-key attributes, the relation is in 2NF and also in 3NF. However, the relation suffers from anomalies (please check it yourself by making the relational instance of the STUDENT5 relation).

The difficulty in this relation is being caused by dependence within the attributes of composite candidate keys.

Definition: A relation is in Boyce-Codd Normal Form (BCNF) if it fulfills the following criteria:

- (i) The relation fulfills the criteria of the Third Normal Form (3NF), and
- (ii) All the determinants (the left side of an FD) are the candidate key.

A relation that is in 3NF and does not have overlapping candidate keys, will also be in BCNF. However, if a 3NF relation have following features then it is NOT in BCNF:

- (a) It has overlapping composite candidate keys.
- (b) The non-overlapping attributes of two candidate keys are functionally dependent.

For example, in the relation STUDENT5, the candidate keys:

(StEnrolNo, CoName) and (StName, CoName) have non-overlapping attributes StEnrolNo and StName. Further, $\text{StEnrolNo} \rightarrow \text{StName}$

The relation STUDENT5 (StEnrolNo, StName, CoNo, CoName) is in 3NF, but not in BCNF.

You can decompose the relation into the following relations using FD

$\text{StEnrolNo} \rightarrow \text{StName}$ as:

STUDENT6(StEnrolNo, CoNo, CoName)
STUDENTNAME (StEnrolNo, StName)

STUDENT6 is still not in BCNF as it has overlapping candidate keys:

(StEnrolNo, CoNo) and (StEnrolNo, CoName) and the FD $\text{CoNo} \rightarrow \text{CoName}$

Thus, STUDENT6 can be decomposed using the FD as above to relations:

STUDENT7 (StEnrolNo, CoNo)
STUDENTNAME (StEnrolNo, StName)
COURSENAME (CoNo, CoName)

The following is another example of BCNF. Consider the relation:

ENROL (StEnrolNo, StName, CoNo, CoName, Dateenrolled)

Let us assume that the relation has the following FDs (We have assumed that CoName are unique):

$$\begin{array}{lcl} \text{StEnrolNo} & \rightarrow & \text{StName} \\ \text{CoNo} & \rightarrow & \text{CoName} \end{array}$$

$$\begin{array}{ccc} \text{CoName} & \xrightarrow{\quad} & \text{CoNo} \\ \text{StEnrolNo, CoNo} & \xrightarrow{\quad} & \text{Dateenrolled} \end{array}$$

The candidate keys of the relation would be:

$$\begin{array}{c} (\text{StEnrolNo}, \text{CoNo}) \\ (\text{StEnrolNo}, \text{CoName}) \end{array}$$

Due to dependency between the non-overlapping attributes of the candidate keys, the 3NF relation ENROL is NOT in BCNF. This relation will have all the stated anomalies (Please draw the relational instance and check these problems). The BCNF decomposition of the relation would be:

$$\begin{array}{c} \text{STUD1 } (\underline{\text{StEnrolNo}}, \text{StName}) \\ \text{COU1 } (\underline{\text{CoNo}}, \text{CoName}) \\ \text{ENROL1 } (\underline{\text{StEnrolNo}}, \underline{\text{CoNo}}, \text{Dateenrolled}) \end{array}$$

You now have a relation that only has information about students, another only about subjects and the third only about relationship enrolls.

Higher Normal Forms:

Are there more normal forms beyond BCNF? Yes, however, these normal forms are not based on the concept of functional dependence. Further normalisation is needed if the relation has multi-valued, join dependencies, etc. These are discussed in Unit 6.

Check Your Progress 2

- 1) Given the following relation of book issue and return in a Library:
BOOKISSUERETURN (student_id, student_name, bookID, book_title, issuedOn, returnedOn)

A student can get many books issued to him/her. Explain anomalies in the relation above with the help of a relational instance.

.....
.....

- 2) Identify the functional dependencies in the relation given in question 1. What are the candidate keys and which of these can be a primary key?

.....
.....

- 3) Normalise the relation of problem 1.

.....
.....
.....
.....

5.6 DESIRABLE PROPERTIES OF DECOMPOSITION

In the previous section, you have learnt the process of normalisation using functional dependency. Normalizing a relation entails decomposing a relation into a number of non-disjoint projections of the relation based on the FDs, so that the three anomalies are minimised. But why are these projections non-disjoint? The non-disjoint relations allow reconstruction of the original relation instance through JOIN operation. In this section, we discuss about the properties of the a good decomposition.

The decomposition of a relation should fulfill the following:

Attribute Preservation: All the attributes of the relation, which is being decomposed, should be part of at least one of the decomposed relations. This is a necessary condition, otherwise the decomposition will be lossy.

Lossless-Join Decomposition: For explaining the concept of lossless-join decomposition, let us first explain a lossy decomposition with the help of an example. This example also explains, why FDs should be used to perform proper decomposition of a relation.

Example: Consider the following instance of a STUDENT9 relation.

STUDENT9 (StEnrolNo, CoNo, Dateenrolled, CoInstructor, InRoom)

With the following set of FDs:

StEnrolNo, CoNo → Dateenrolled

CoNo → CoInstructor

CoInstructor → InRoom

Suppose you decompose the STUDENT9 relation randomly into two relations ST1

and ST2 as follows:

ST1 (StEnrolNo, CoNo, Dateenrolled, CoInstructor)

ST2 (StEnrolNo, InRoom)

Are there problems with this decomposition? Yes, but for the time being, let us focus on the question, whether this decomposition is lossless? Consider the following instance of the relation:

STUDENT9

StEnrolNo	CoNo	Dateenrolled	CoInstructor	InRoom
1001	MCS-201	01-02-2022	Preeti	1
1001	MCS-202	01-02-2022	Salim	2
1002	MCS-201	13-02-2022	Preeti	1
1002	MCS-203	13-02-2022	Shashi	3
1003	MCS-202	15-02-2022	Salim	2

Figure 5.7: An instance of STUDENT9 relation

The decomposed relations ST1 and ST2 would be:

ST1

StEnrolNo	CoNo	Dateenrolled	CoInstructor
1001	MCS-201	01-02-2022	Preeti
1001	MCS-202	01-02-2022	Salim
1002	MCS-201	13-02-2022	Preeti
1002	MCS-203	13-02-2022	Shashi
1003	MCS-202	15-02-2022	Salim

ST2

StEnrolNo	InRoom
1001	1
1001	2
1002	1
1002	3
1003	2

Will you be able to reconstruct the original instance of relation using ST1 and ST2? For this simply take a JOIN of the two relations ST1 and ST2 on StEnrolNo, which is the only common attribute. The joined instance would be:

ST1JOINST2

StEnrolNo	CoNo	Dateenrolled	CoInstructor	InRoom
1001	MCS-201	01-02-2022	Preeti	1
1001	MCS-201	01-02-2022	Preeti	2
1001	MCS-202	01-02-2022	Salim	1
1001	MCS-202	01-02-2022	Salim	2
1002	MCS-201	13-02-2022	Preeti	1
1002	MCS-201	13-02-2022	Preeti	3
1002	MCS-203	13-02-2022	Shashi	1

1002	MCS-203	13-02-2022	Shashi	3
1003	MCS-202	15-02-2022	Salim	2

Thus, the resulting relation obtained is not the same as that of *Figure 5.7*. The joined relation contains a number of spurious tuples that were not in the original relation. Because of these additional tuples, you have lost the information, such as the instructor Preeti is located in Room number 1. Such decompositions are called **lossy decompositions**. A lossless join decomposition guarantees that the join will result in exactly the same relation as was decomposed. One might think that there might be other ways of recovering the original relation from the decomposed relations but, sadly, no other operators can recover the original relation if the join does not (why?). You need to analyse why the decomposition is lossy. The common attribute in the above decompositions was StEnrolNo. The common attribute is the glue that gives us the ability to find the relationships between different relations by joining the relations together. *If the common attribute has been the primary key of at least one of the two decomposed relations, the problem of losing information would not have existed.* You may use FDs to decompose the relation STUDENT9 into 2NF and then to 3NF, to produce the following relational instance:

STENROL (StEnrolNo, CoNo, Dateenrolled)
using the FD StEnrolNo, CoNo → Dateenrolled

STENROL

StEnrolNo	CoNo	Dateenrolled
1001	MCS-201	01-02-2022
1001	MCS-202	01-02-2022
1002	MCS-201	13-02-2022
1002	MCS-203	13-02-2022
1003	MCS-202	15-02-2022

COUINST (CoNo, CoInstructor) using the FD CoNo → CoInstructor

COUINST

CoNo	CoInstructor
MCS-201	Preeti
MCS-202	Salim
MCS-203	Shashi

INSROOM (CoInstructor, InRoom) using the FD CoInstructor → InRoom

INSROOM

CoInstructor	InRoom
Preeti	1
Salim	2

You may please join the three relations to check that that the decomposition is lossless-join decomposition. *The dependency-based decomposition scheme as discussed in the section 5.5 creates lossless decomposition.*

Dependency Preservation: It is clear that the decomposition must be lossless so that you do not lose any information from the relation that is decomposed. Dependency preservation is another important requirement since a *dependency is a constraint* on the database. If all the attributes appearing on the left and the right side of a dependency appear in the same relation, then a dependency is considered to be preserved. Thus, dependency preservation can be checked easily. Dependency preservation is important, because as stated earlier, dependency is a constraint on a relation. Thus, if a constraint is split over more than one relation (dependency is not preserved), the constraint would be difficult to meet. We will not discuss this in more detail in this unit. You may refer to the further readings for more details. However, let us state some basic points:

"Normalisation to 3NF is lossless decomposition. It also preserves the dependencies."

"Normalisation to BCNF is lossless decomposition. However, it may not preserve dependencies."

Reduction of Redundancy: Redundancy is the cause of anomalies. Normalisation results in reduction of redundancy.

5.6 RULES OF DATA NORMALISATION

In the previous sections, you have gone through various normal forms. In this section, the normalisation using FDs is presented as a set of practical rules:

1. Identify and eliminate Attributes with multiple data values into separate relations.
2. Identify and eliminate those attributes, which are not the part of any key attributes but are dependent on the part of the composite primary key.
3. Identify and eliminate those non-key attributes, which are dependent on other non-key attributes.
4. Identify and eliminate non-overlapping composite candidate key attributes, which are dependent on each other.

Let's explain these steps of Normalisation through an example. Let us make a list of all the employees in the company. In the original employee list, each employee name is followed by any databases that the employee has experience with. Some might know many, and others might not know any.

Employee ID (empid)	Employee Name (empname)	DBMS Known (DBMS)	Department (dept)	Department Location (loc)
101	Gurmeet	MySQL	Quality Assurance	Mumbai
102	Hanif	DB2	Database Design	Delhi
103	Manish	Oracle, PostgreSQL	Frontend Design	Hyderabad
104	Sameer Singh	SQLserver, Oracle	Database Design	Delhi

Figure 5.8: Table of Data Not in 1NF

For attributes Department and Department Location will be considered in Step-3.

5.6.1 Eliminate Repeating Groups

Please observe that Figure 5.8 has a repeating group – DBMS Known. The problem with such repeating groups can be explained with the help of a query. Consider a query - “Find the list of employees, who know Oracle”.

To answer this query, you need to perform an awkward scan of the entire list of attributes Database-Known, looking for references to DB2. This is inefficient and an extremely untidy way to retrieve information.

You may convert the relation to 1NF (For the following discussion, we have ignored two attributes - Department and Department Location. These two attributes will be part of Employee relation). The two decomposed relations on eliminating the repeating groups are shown in Figure 5.9. The elimination of repeating group DBMS Known from the Employee relation, which has primary key - Employee ID, leaves the Employee relation in 1NF. However, this elimination requires that you add another relation, which can store information about the DBMS known by each employee. This new relation is named DBMSknown. It has three attributes, including a foreign key for relating the two relations with a JOIN operation. Now, you can answer the question by looking in the database relation for "Oracle" and getting the list of Employees. Please note that although the name of the DBMSs are unique, yet we have added a DBMS code field in the decomposed relation (Figure 5.8):

EMPLOYEE			
<u>empid</u>	<u>empname</u>	<u>dept</u>	<u>loc</u>
101	Gurmeet	Quality Assurance	Mumbai
102	Hanif	Database Design	Delhi
103	Manish	Frontend Design	Hyderabad
104	Sameer Singh	Database Design	Delhi

DBMSknown		
<u>empid</u>	<u>DBMScode</u>	<u>DBMS</u>
101	1	MySQL
102	2	DB2
103	3	Oracle
103	4	PostgreSQL
104	5	SQLserver
104	3	Oracle

Figure 5.9: 1NF relation after Eliminating Repeating Groups

5.6.2 Eliminate Redundant Data

In the “DBMSknown” relation above, the primary key is made up of the *empid* and the *DBMScode*. The attribute *DBMS* depends only on the *DBMScode*. The same *DBMS* will appear redundantly every time its associated *DBMScode* appears in the *DBMSknown* Relation. Thus, this database relation has redundancy, for example, *DBMScode* value 3 is Oracle, which is repeated twice. In addition, it also suffers insertion anomaly that is you cannot enter Sybase in the relation as no employee has that database skill.

The deletion anomaly also exists. For example, if you remove an employee with *empid* 3; no employee has a skill in PostgreSQL and the information that *DBMScode* 4 is the code of PostgreSQL will be lost.

To avoid these problems, you need a second normal form. To achieve this, you isolate the attributes that depend on the entire composite key from the attributes that depend only on the *DBMScode*. This results in decomposition of *DBMSknown* relation into two relations: “DBMSlist” and “EMPDBMS” which lists the databases for each employee. The EMPLOYEE relation is already in 2NF as all the *empid* determines all other attributes.

EMPDBMS	
<u>empid</u>	<u>DBMScode</u>
101	1
102	2

DBMSlist	
<u>DBMScode</u>	<u>DBMS</u>
1	MySQL
2	DB2

103	3
103	4
104	5
104	3

3	Oracle
4	PostgreSQL
5	SQLserver

5.6.3 Eliminate Columns Not Dependent on Key

The Employee Relation satisfies -

First normal form - As it contains no repeating groups.

Second normal form - As it does not have a multi-attribute key.

Now, let us add the remaining two attributes of Employee relation. The employee relation is in 2NF but not 3NF. Why?

EMPLOYEE			
<u>empid</u>	empname	dept	loc
101	Gurmeet	Quality Assurance	Mumbai
102	Hanif	Database Design	Delhi
103	Manish	Frontend Design	Hyderabad
104	Sameer Singh	Database Design	Delhi

The key to the Employee relation is *empid*. The attribute *loc* describes information about the Department and not about an Employee. To achieve the third normal form, *dept* and *loc* must be moved into a separate relation. Since they describe a department, thus, the attribute *dept* becomes the key of the new “Department” relation.

The motivation for this decomposition is that you want to avoid update, insertion and deletion anomalies.

EMPLOYEElist		
<u>empid</u>	empname	dept
101	Gurmeet	Quality Assurance
102	Hanif	Database Design
103	Manish	Frontend Design
104	Sameer Singh	Database Design

DEPARTMENT Relation		
<u>deptid</u>	dept	loc
1	Quality Assurance	Mumbai
2	Database Design	Delhi
3	Frontend Design	Hyderabad

You may use these steps for decomposition till BCNF.

Check Your Progress 3

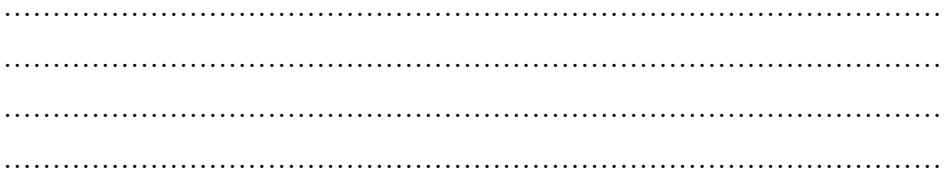
- 1) Why should functional dependencies be preserved in decomposition?

.....

- 2) Explain the term lossless-join decomposition.

.....

- 3) List various steps that may be followed to decompose a relation up to BCNF.



5.7 SUMMARY

This unit discusses some of the most important aspects of a good database design. The unit first defines the concept of referential integrity constraint and entity integrity constraints. It then discusses different forms of normalisation based on the concept of function dependency. A functional dependency highlights the relationships among the attributes of a relation. Different dependency among attributes leads to the problems of update anomalies, insertion anomalies and deletion anomalies. Different forms of normalisation decompose relations, using the FDs, to remove anomalies. The concept of FD is used for the process of normalisation.

This unit also defines the features of a good decomposition of a relation. The most important being attribute preservation, dependency preservation and lossless-join decomposition. Finally, the unit summarises the rules of normalisation with the help of an example.

5.10 SOLUTIONS/ANSWERS

Check Your Progress 1

- 1) The following table shows the candidate keys and primary keys of the relations:

Relation	Candidate Keys	Primary key
S (Supplier)	SNO, SNAME*	SNO
P (Part)	PNO, PNAME*	PNO
J (Project)	PROJ NO, JNAME*	PROJNO
SPJ	(SNO+PNO+PROJNO)	(SNO+PNO+PROJNO)

* Only if the values are assumed to be unique, this may be incorrect for large systems.

- 2) SNO in S, PNO in P, PROJNO in J and (SNO+PNO+PROJNO) in SPJ should not contain NULL value. Also, no part of the primary key of SPJ, that is SNO or PNO or PROJNO should contain NULL value.
- 3) Foreign keys exist only in SPJ, where SNO references SNO of S; PNO references PNO of P; and PROJNO references PROJNO of J. The referential integrity necessitates that all matching foreign key values must exist.
- 4) You may select the following referential actions:

For operations like Delete and update, you should use referential action as RESTRICT. This selection would restrict loss of information from the SPJ relation, in case you delete a tuple in S or P or J relation. Insertion of records does not create a problem provided the referential integrity constraints are met.

Check Your Progress 2

- 1) The database suffers from all the anomalies; let us demonstrate these with the help of the following relation instance or state of the relation:

Database Integrity and Normalisation

student_id	Student_name	bookID	Book_title	issuedOn	returnedOn
A 101	Abishek	0050	DBMS	15/01/05	25/01/05
R 102	Raman	0125	DS	25/01/05	29/01/05
A 101	Abishek	0060	Multimedia	20/01/05	NULL
R 102	Raman	0050	DBMS	28/01/05	NULL

Is there any data redundancy?

Yes, the information is getting repeated about student names and book Title. This may lead to an update anomaly in case of changes made to the data value of the book. In addition, the library may be having many more books that have not been issued yet. The information of such books cannot be added to the relation as the primary key to the relation is: (student_id + bookID + issuedOn). (This would involve the assumption that the same book can be issued to the same student only once in a day). Thus, you cannot enter the information about bookID and book_title of those books, which has not been issued to a student. This is an insertion anomaly. Similarly, you cannot enter student_id if a book is not issued to that student. This is also an insertion anomaly. As far as the deletion anomaly is concerned, suppose Abishek did not collect the Multimedia book, so this record needs to be deleted from the relation (tuple 3). This deletion will also remove the information about the Multimedia book that is its bookID and book_title. This is a deletion anomaly for the given instance.

- 2) The FDs of the relation are:

$$\text{student_id} \rightarrow \text{student_name}$$

(1)

$$\text{bookID} \rightarrow \text{book_title}$$

(2)

$$\text{bookID, student_id, issuedOn} \rightarrow \text{returnedOn}$$

(3)

Why is the attribute issuedOn on the left hand of the FD above? Because a student, for example, Abishek can be issued the book having the bookID 0050 again on a later date, let say in February after Raman has returned it. Also note that returnedOn may have NULL value, but that is allowed in the FD. FD only necessitates that the value may be NULL or any specific data that is consistent across the instance of the relation.

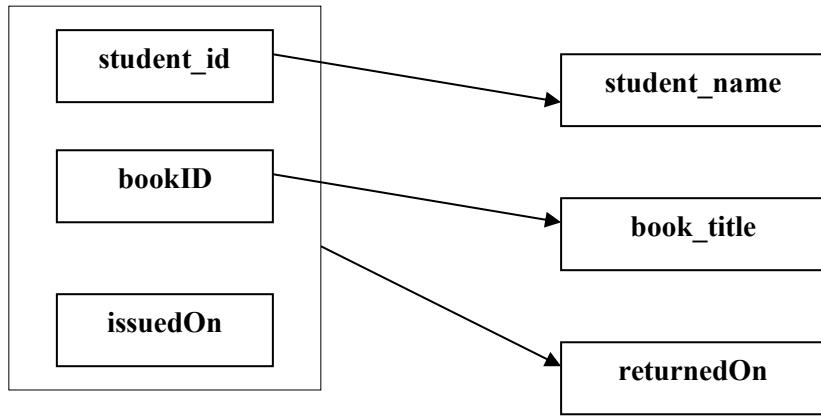
Just one candidate key, which is also chosen as the primary key, is: bookID, student_id, issuedOn.

Some interesting domain and procedural constraints in this database are:

- A book cannot be issued again unless it is returned.
- A book can be returned only after the date on which it has been issued.
- A student can be issued a limited/maximum number of books at a time.

You will study about these constraints later in the Block.

- 3) The relation is in 1NF. The following is a FD diagram for the relation:



1NF to 2NF:

The composite key to the relation is (student_id + bookID). The attributes student_name depends on student_id, which is part of the composite key, likewise book_title attributes is dependent on bookID attribute, which is part of the composite key so the decomposition based on following FDs would be:

[Reason: FD (1)]

[Reason: FD (2)]

[Reason: FD (3)]

2NF to 3 NF and BCNF:

All the relations are in 3NF and BCNF also. As there is no dependence among non-key attributes and there are no overlapping candidate keys.

Please note that the decomposed relations have no anomalies. Let us map the relation instance here:

MEMBER

Member - ID	Members Name
A 101	Abhishek
R 102	Raman

BOOK

Book - Code	Book – Name
0050	DBMS
0060	Multimedia
0125	OS

ISSUE_RETURN

Member - ID	Book - Code	Issue - Date	Return – Date
A 101	0050	15/01/05	25/01/05
R 102	0125	25/01/05	29/01/05
A 101	0060	20/01/05	NULL
R 102	0050	28/01/05	NULL

- i) There is no redundancy, so no update anomaly is possible in the relations above.
- ii) The insertion of new book and new student can be done in the BOOK and MEMBER tables respectively without any issue of book to student or vice versa. So, no insertion anomaly.
- iii) Even on deleting record 3 from ISSUE_RETURN it will not delete the information the book 0060 titled as Multimedia as this information is in separate table. So, no deletion anomaly.

Check Your Progress 3

- 1) Dependency preservation within a relation helps in enforcing constraints that are implied by dependency over a single relation. In case, you do not preserve dependency then constraints might be enforced on more than one relation that is quite troublesome and time consuming.
- 2) A lossless join decomposition is one in which you can reconstruct the original table without loss of information that means exactly the same tuples are obtained on taking join of the relations obtained on decomposition. Lossless join decomposition requires that decomposition should be carried out on the basis of FDs.
- 3) The steps of Normalisation are:
 1. Remove repeating groups of each of the multi-valued attributes.
 2. Then remove redundant data and its dependence on part keys.
 3. Remove columns from the table that are not dependent on the key, that is remove transitive dependency.
 4. Check if there are overlapping composite candidate keys. If yes, check for any dependency among the non-overlapping attributes of the composite candidate keys; and remove such dependency.



UNIT 6 HIGHER NORMAL FORMS

Structure

- 6.0 Introduction
 - 6.1 Objectives
 - 6.2 Multivalued Dependency
 - 6.3 Fourth Normal Form (4NF)
 - 6.4 Join Dependency
 - 6.5 5NF
 - 6.6 Other Normal Forms
 - 6.7 Summary
 - 6.8 Solutions/Answers
-

6.0 INTRODUCTION

In the previous unit of this block, you have gone through the concept of single-valued dependency, called functional dependency (FD). Further, the previous Unit discussed different forms of normalisations that can be arrived at by removing different types of single-valued dependency, viz. 1NF, 2NF, 3NF and BCNF. However, relational databases may have several other types of dependencies, which results in redundancy in a relation. Such dependencies, thus causes the update anomaly, insertion anomaly and deletion anomaly in a relation. Therefore, more normal forms have been designed to address this problem.

This Unit focuses on the higher normal forms, namely fourth normal form (4NF), which is based on the concept of multivalued dependency (MVD); and fifth normal form (5NF), which is based on the concept of Join dependency. The unit also introduces you to other normal forms. For more details on other normal forms, you may refer to the further readings. In general, these higher normal forms are not very popular among industries, however they propose good theoretical perspectives for the database design.

6.1 OBJECTIVES

After going through this unit, you should be able to:

- Explain the concept of multivalued dependencies;
 - Perform normalisation up to 4NF;
 - Explain the join dependency;
 - Explain the 5NF;
 - Define other normal forms.
-

6.2 MULTIVALUED DEPENDENCIES

An attribute in a relational model represents only a single value information. How will you represent the information if a particular attribute is multivalued? Such multivalued attributes would require that

information of all other attributes is repeated in an instance of the relation. This will result in multiple tuples in a single relation to represent information about one entity. The primary key of such a relation would be a composite key involving the primary key of the entity and the multivalued attribute. This situation becomes worse, if an entity has more than one multivalued attribute. Such an entity will be represented by several tuples. The following example explains this situation:

Example 1: Let us consider a relation ‘employee’.

emp (e#, project, tool)

An employee can work on several projects and an employee has skills in several tools, therefore, there is no functional dependency in this relation. However, this relation has two multivalued attributes: *project* and *tool*.

The attributes *project* and *tool* are assumed to be independent of each other, as any project can use any tool. The following table (not relation) defines this situation:

e# (Employee Number)	Project	Tool
E001	DBMS, Ecommerce	Python, Virtualization
E002	Ecommerce, Bank Automation	PostgreSQL, Virtualization
E003	Bank Automation	PostgreSQL

Figure 6.1: Sample Data

However, to represent this information through 1NF, you need to create the following relation:

e#	project	tool
E001	DBMS	Python
E001	DBMS	Virtualization
E001	Ecommerce	Python
E001	Ecommerce	Virtualization
E002	Ecommerce	PostgreSQL
E002	Ecommerce	Virtualization
E002	Bank Automation	PostgreSQL
E002	Bank Automation	Virtualization
E003	Bank Automation	PostgreSQL

Figure 6.2: 1NF relation of data of Figure 6.1

This relation has no FD and primary key of the relation is composite key (*e#, project, tool*), therefore, the relation is in 3NF and BCNF. However, it suffers from the problem of redundancy, for example, the employee E001 is working on DBMS is appearing twice, so is that E001 knows the tool Python. This may lead to update anomaly. Further, you cannot insert information about a new employee, who knows the tools PostgreSQL, but has not been assigned to any project. In addition, if you remove the employee E003 from the Bank Automation project, the information that E003 has skill in PostgreSQL would be lost. Thus, the relation suffers from update, insertion and deletion anomalies.

How can you now normalise the relation, as this contains no FD? For addressing the issues related to such relations, we may first define the concept of multivalued dependency, which relates to relations that contain more than one multivalued attribute. Let us define the multivalued dependency formally for a relation $R(X, Y, Z)$, where X, Y and Z are a set of attributes.

Multivalued dependency (MVD): Given a relation $R(X, Y, Z)$, a MVD $X \rightarrow\rightarrow Y$ will hold in relation R if for a specific value of attribute set X , there is an associated set of values of attribute set Y , such that these

multiple-associated (zero or more) values of attributes Y depends only on the value of attribute X. Further, values of attribute set Y have no dependence on the value of attribute set Z.

Hence, if $MVD X \rightarrow\rightarrow Y$ holds in R , another $MVD X \rightarrow\rightarrow Z$ would also hold, as the function of the attribute set Y and attribute set Z is symmetrical.

In the *Example 1* given above, the employee number can determine all the projects, employee is working on, and also an employee number can determine all the tools in which that employee is skillful. Further, both the *project* and *tool* attributes are independent of each other. Therefore, the following MVDs hold in relation of example 1:

$$\begin{aligned} e\# &\rightarrow\rightarrow project \\ e\# &\rightarrow\rightarrow tool \end{aligned}$$

Let us now define the concept of MVD in a different way. Consider the relation $R(X, Y, Z)$ having a multi-valued set of attributes Y associated with a value of X . Assume that the attributes Y and Z are independent, and Z is also multi-valued. Now, more formally, $X \rightarrow\rightarrow Y$ is said to hold for $R(X, Y, Z)$ if $t1$ and $t2$ are two tuples in R that have the same values for attributes X ($t1[X] = t2[X]$) then R also contains tuples $t3$ and $t4$ (not necessarily distinct) such that:

$$\begin{aligned} t1[X] &= t2[X] = t3[X] = t4[X] \\ t3[Y] &= t1[Y] \text{ and } t3[Z] = t2[Z] \\ t4[Y] &= t2[Y] \text{ and } t4[Z] = t1[Z] \end{aligned}$$

For example, consider Figure 6.2, the two such tuples are:

Tuple 1: E001	DBMS	Python
Tuple 4: E001	Ecommerce	Virtualization

Then two more tuples must exist as follows:

E001	DBMS	Virtualization
E001	Ecommerce	Python

Please check these two are Tuple 2 and Tuple 3 in Figure 6.2

We are, therefore, requiring that every value of Y appears with every value of Z to keep the relation instances consistent. In other words, the above conditions insist that Y and Z are determined by X alone and there is no relationship between Y and Z , since Y and Z appear in every possible pair and hence these pairings present no information and are of no significance. Only if some of these pairings were not present, there would be some significance in the pairings.

(Note: If Z is single-valued and functionally dependent on X then $Z1 = Z2$. If Z is multivalued dependent on X then $Z1 \neq Z2$).

The theory of multivalued dependencies is very similar to that for functional dependencies. Given a set of MVDs, say D , you can find D^+ , the closure of D using a set of axioms. We do not discuss the axioms here. You may refer to this topic in further readings.

Before explaining the 4NF of a relation, one more term needs to be defined. It is the Trivial MVD, which is defined next.

Trivial MVD: A MVC $X \rightarrow\rightarrow Y$ is called trivial MVD if either Y is a subset of X or X and Y together form the relation R .

The MVD is trivial since it results in no constraints being placed on the relation. For example, consider a relation *dependent* ($e\#$, $e_{dependent}\#$). An employee can have zero or more dependents, therefore, $e\#$ uniquely determines the **values** of $e_{dependent}\#$. However, this MVD is trivial, as $e\#$, $e_{dependent}\#$ together forms the relation *dependent*. This *dependent* relation cannot be decomposed any further.

Therefore, a relation having non-trivial MVDs must have at least three attributes; two of them would be multivalued and not dependent on each other. Non-trivial MVDs result in the relation having some constraints on it since all possible combinations of the multivalued attributes are then required to be present in the relation. In the next section, we discuss how MVD can be used to decompose a relation into fourth normal form.

6.3 FOURTH NORMAL FORM

In this section, first we define the fourth normal form (4NF) and then present an example about how MVD can be used to decompose a relation to 4NF.

A relation R is in 4NF if for all the multivalued dependencies ($X \rightarrow\rightarrow Y$) any one of the following clauses holds:

- the multivalued dependencies ($X \rightarrow\rightarrow Y$) is trivial,
- X is a candidate key for R .

The dependency $X \rightarrow\rightarrow \emptyset$ or $X \rightarrow\rightarrow Y$ in a relation R (X, Y) is trivial, since they must hold for all R (X, Y). In this case, R (X, Y) is in 4NF. Similarly, if in a relations R (A, B, C) with only three attributes, if a trivial MVD $(A, B) \rightarrow\rightarrow C$ holds, then R (A, B, C) is in 4NF.

If a relation has more than one multivalued attribute, you should decompose it into the fourth normal form using the following rules of decomposition:

For a relation $R(X, Y, Z)$, if it contains two nontrivial MVDs $X \rightarrow\rightarrow Y$ and $X \rightarrow\rightarrow Z$, then decompose the relation into $R_1(X, Y)$ and $R_2(X, Z)$ or more specifically, if there holds a non-trivial MVD in a relation R (X, Y, Z) of the form $X \rightarrow\rightarrow Y$, such that $X \cap Y = \emptyset$, that is the set of attributes X and Y are disjoint, then R must be decomposed to $R_1(X, Y)$ and $R_2(X, Z)$, where Z represents all attributes other than those in X and Y .

Intuitively R is in 4NF if

- (1) All dependencies are a result of keys.
- (2) When multivalued dependencies exist, a relation should not contain two or more independent multivalued attributes.

The decomposition of a relation to achieve 4NF would normally result in not only reduction of redundancies but also avoidance of anomalies.

Example 2: Normalise the relation *emp* (*e#*, *project*, *tool*) shown in Figure 6.2 using the MVDs:

$e\# \rightarrow\rightarrow project$ and $e\# \rightarrow\rightarrow tool$.

The relation has no FD, but two MVDs, therefore, the relation is in BCNF but not 4NF. To decompose the relation into 4NF, you may use any of the MVD. The decomposed relation would be:

empproj (*e#*, *project*) with MVD $e\# \rightarrow\rightarrow project$, which is now a trivial MVD; and

empproj (e#, project) with MVD $e\# \rightarrow\!\!\!\rightarrow$ tool, which is a trivial MVD.

On decomposition of the relation in Figure 6.2, by taking the projections as stated above, the relational state of the decomposed relations would be:

<i>empproj</i>		<i>emptool</i>	
<u>e#</u>	<u>project</u>	<u>e#</u>	<u>tool</u>
E001	DBMS	E001	Python
E001	Ecommerce	E001	Virtualization
E002	Ecommerce	E002	PostgreSQL
E002	Bank Automation	E002	Virtualization
E003	Bank Automation	E003	PostgreSQL

Figure 6.3: Decomposition of *emp (e#, project, tool)* relation to 4NF

You can observe the following in Figure 6.3

- The key to relations empproj and emptool are (e#, project) and (e#, tool) respectively.
- There is no redundancy in empproj and emptool relations.
- Both the relations do not suffer from any anomaly.

So far, you are able to decompose a relation based on FD and MVD. Are there any other forms of dependencies? Researchers have shown several kinds of dependencies. However, for this course we will discuss about one more type of dependency, called the join dependency, which is discussed next.

☞ Check Your Progress 1

- 1) What are Multi-valued Dependencies? When can you say that a constraint X is multi-valued dependency?

.....
.....
.....

- 2) Identify the MVDs in the following relation. Decompose the relation into 4NF using the MVDs.

EMP

ENAME	PNAME	DNAME
Rohan	Big Data	AI Unit
Rohan	Machine Learning	Analytics
Rohan	Big Data	Analytics
Rohan	Machine Learning	AI Unit

.....
.....
.....

- 3) Convert the following relation to 4NF relations.

SUPPLY

SNAME	PARTNAME	PROJNAME
XYZ Pvt Ltd	Bolt	Big Data
XYZ Pvt Ltd	Nut	Machine Learning
ABC Ltd	Bolt	Machine Learning
Info Comm Ltd	Nut	AI
ABC Ltd	Nail	Big Data
ABC Ltd	Bolt	Big Data
XYZ Pvt Ltd	Bolt	Machine Learning

.....
.....
.....

6.4 JOIN DEPENDENCIES

As discussed in the previous section, a relation that suffers from insertion, update and deletion anomalies is decomposed using either FD or MVD. The normal forms require that a given relation R , if not in the given normal form, should be decomposed in two relations to meet the requirements of the normal form. However, in some cases, a relation can have problems like redundancy leading to anomalies, yet it cannot be decomposed in two relations without loss of information. In such cases, it may be possible to decompose the relation in three or more relations. When does such a situation arise? Such cases normally happen when a relation has at least three attributes such that all those values are totally independent of each other. It may also be the case when a ternary relationship exists among three entities.

Following example explains this in detail. Consider a relation:

ProjToolEmp (projectid, toolid, empid)

There are no constraints on this relation that is:

- Any project can use any tools
- Any tool can be used by any employee
- Any employee can work on any project
- No employee would use all the tools
- No employee would work on all the projects
- No project uses all the tools
- All three attributes are independent of each other

Assume that the relation has the following relational instance:

Tuple#	projectid	toolid	empid
1	P1	T1	E1
2	P2	T2	E2
3	P1	T2	E2

Figure 6.4: A ternary relation with all independent attributes

The relation above does not have any FDs and MVDs since the attributes projectid, toolid and empid are independent; they are related to each other only by the pairings that have significant information in them.

For example, the first tuple of relation states that employee E1 works on P1 project, which uses tool T1. The key to the relation is the composite key (projectid, toolid, empid). The relation is in 4NF, but still suffers from the insertion, deletion, and update anomalies, as the information that Employee E1 can use tool T1 is redundant in tuple 3. Similarly, information like project P3 uses tool T1 cannot be inserted in the relation, as no employee has started working on the project. Likewise, on deleting tuple 2 from the relation, may delete the information like Employee E2 can use tool T2. Therefore, you need to decompose the relation to minimise the anomalies. As there are no FDs or MVDs in this relation, there is no basis of decomposition. You may try to see what happens when you decompose the relation into the following two relations:

ProjectTool	
projectid	toolid
P1	T1
P2	T2
P1	T2

ProjectEmployee	
projectid	empid
P1	E1
P2	E2
P1	E2

Figure 6.5: A decomposition of ternary relation of Figure 6.4

What happens when you take Join of the two relations on the attribute *projectid*:

Tuple#	projectid	toolid	empid
1	P1	T1	E1
2	P1	T1	E2
3	P2	T2	E2
4	P1	T2	E1
5	P1	T2	E2

Figure 6.6: ProjectTool JOIN ProjectEmployee

You may notice that the tuples 2 and tuple 4 in the relation state in Figure 6.6 were not existent in original relational state given in Figure 6.4. Thus, this decomposition is a lossy decomposition. So, what should be done to remove the anomalies?

In such situation, you may have to decompose the relation into three relations. Two of which are already shown in Figure 6.5. A third relation as shown in Figure 6.7 must be added.

ToolEmployee	
toolid	empid
T1	E1
T2	E2

Figure 6.7: The third relation

In order to obtain the original relation, you need to perform the following join operation:

ProjectTool JOIN ProjectEmployee JOIN ToolEmployee

Figure 6.6 represents (*ProjectTool JOIN ProjectEmployee*) which can be joined with *ToolEmployee*

Tuple#	projectid	toolid	empid
1	P1	T1	E1
2	P1	T1	E2
2	P2	T2	E2

<u>4</u>	<u>P1</u>	<u>T2</u>	<u>E1</u>
3	P1	T2	E2

Figure 6.8: ProjectTool JOIN ProjectEmployee JOIN ToolEmployee

Please note that Tuple 2 and Tuple 4 of Figure 6.6 will not be part of Figure 6.8, as there are no matching tuples in Figure 6.7. Thus, you can observe that Figure 6.4 and Figure 6.8 are identical.

Therefore, the relation ProjToolEmp (projectid, toolid, empid), which has no FD and MVD can be decomposed into three relations, which can be joined to form the original relation. This forms the concept of join dependency, which is explained next.

Join Dependency (JD): A relation R satisfies join dependency over the projections (P_1, P_2, \dots, P_n) if and only if every instance of R, the join of P_1, P_2, \dots, P_n creates the instance of R.

For example, the instance of ternary relation ProjToolEmp (projectid, toolid, empid), as shown in Figure 6.4, when decomposed to two relations, as shown in Figure 6.5, does not satisfy the join dependency. This is shown in Figure 6.6. On addition of the third relation, as shown in Figure 6.7, the three projections ProjectTool (projectid, toolid), ProjectEmployee(projectid, empid) and ToolEmployee (toolid, empid) satisfies the join dependency as JOIN on the three projections, viz. ProjectTool (projectid, toolid), ProjectEmployee(projectid, empid) and ToolEmployee (toolid, empid), is the same as the instance of the relation. Thus, the three projections, as stated above, satisfies the join dependency.

6.4 FIFTH NORMAL FORM

The fifth normal form (5NF) (Project-Join normal form (PJNF)) deals with join-dependencies, which is a generalisation of the MVD. The aim of the fifth normal form is to have relations that cannot be decomposed further. A relation in 5NF cannot be constructed from several smaller relations.

A relation R is in 5NF if for all the join dependencies any one of the following clauses holds:

- (a) *join-dependency (P_1, P_2, \dots, P_n) is trivial (that is, one of the P_i 's is R)*
- (b) *Every P_i is a super key of R.*

For example, the instance of ternary relation ProjToolEmp (projectid, toolid, empid), as shown in Figure 6.4, has a Join Dependency (ProjectTool (projectid, toolid), ProjectEmployee(projectid, empid), ToolEmployee (toolid, empid)). Therefore, this relation is not in 5NF, as it violates the clauses of the 5NF, given above. You can decompose the relation ProjToolEmp (projectid, toolid, empid) into its projections as follows:

ProjectTool (projectid, toolid)
 ProjectEmployee(projectid, empid)
 ToolEmployee(toolid, empid)

Each of these three relations are in 5NF, as they have trivial join dependency. The instance of each of the decomposed relations is shown in Figure 6.5 and Figure 6.7. You may also observe that this decomposition is lossless join decomposition. It may be noted that every MVD is also a join dependency. Therefore, every PJNF (5NF) schema is also in 4NF. A relation schema that has Join Dependencies and suffers from anomalies, can be decomposed into projections, as per the join dependency. The new relations would be in PJNF schema.

Check Your Progress 2

1) What is join dependency?

.....
.....

2) Define 5NF.

.....
.....

3) Convert the relational instance of SUPPLY relation, as given in question 3 of check your progress 1. to 5NF.

.....
.....
.....

6.5 OTHER NORMAL FORMS

The researchers of database systems have found additional dependencies and normal forms. This section introduces the basic concept behind these forms. First, we define some additional types of dependencies.

Inclusion Dependency

The inclusion dependency has been designed for two specific types of database constraints that are not defined by the concept of FD, MVD and Join dependency. These two constraints are:

- Foreign key constraints
- Class/subclass relationships

Please note that these constraints are between two relations. Therefore, it requires a new form of formal definition. We define it in the context of foreign key constraints.

Consider two relations R1 and R2, which are related through a foreign key constraint such that a set of attributes in R1, say X, is the foreign key that references the relation R2 on a set of attributes, say Y. Please note that attribute sets X and Y must have similar number of attributes and similar domains, so that foreign key constraint is applicable. The inclusion dependency for such a situation will be defined as follows:

An **inclusion dependency** (denoted as $R1.X < R2.Y$) if the following relationship between the projections holds:

$$\pi_X(r1) \subseteq \pi_Y(r2) \tag{1}$$

Where r1 and r2 are the instances of relation R1 and R2 respectively at the same instance of time.

For example, consider the following two relational instances:

Relation: DEPT

deptID	deptName	deptlocation
D01	Marketing	Delhi
D02	Production	Mumbai
D03	HR	Delhi

Relation: EMP

empID	empName	salary	department
E01	ABC	30000	D01
E02	BCD	40000	D01
E03	CDE	45000	D02
E04	EFG	35000	D02

There exists a foreign key between the two relations, viz. department in EMP relation references deptID in DEPT relation. Therefore, the inclusion dependency $EMP.department < DEPT.deptID$ must hold for the given relational instances.

The equation (1) for this may be (assuming that relational instance of DEPT is dept and EMP is emp):

$\pi_X(r1)$ is $\pi_{department}(emp)$, which would be:

emp
department
D01
D02

and $\pi_Y(r2)$ is $\pi_{deptID}(dept)$, which would be:

dept
deptID
D01
D02
D03

You may observe that the inclusion dependency $EMP.department < DEPT.deptID$ holds, as $\pi_{department}(emp) \subseteq \pi_{deptID}(dept)$.

Template Dependency

The template dependency is specified in the form of a template and can be used to represent any generic dependency. These dependencies can define any constraints in the form of a template. A template consists of two parts – the first part which shows the tuples that may exist in a relation is called the hypotheses, which are followed by the conclusion of the template.

A template dependency can be used to represent any dependency in this form. The following example shows how a FD can be represented using a template dependency.

Example: Consider a relation RESULT (studentid, courseid, grade) with the FD

studentid, courseid → grade

Represent this FD using template dependency.

Description	<u>studentid</u>	<u>courseid</u>	grade
Hypothesis	S1	C1	G1

	S1	C1	G2
Conclusion	G1 = G2		

The following example shows, how MVDs can be represented using template dependency.

Example: Consider a relation EMPLOYEE (e#, project, tool) with the set of MVDs

$e\# \rightarrow\!\!\! \rightarrow$ project and $e\# \rightarrow\!\!\! \rightarrow$ tool.

The following template dependency defines these MVDs:

Description	<u>e#</u>	project	tool
Hypothesis	E1	P1	T1
	E1	P2	T2
Conclusion	E1	P1	T2
	E1	P2	T1

One example of this MVD is shown with attribute values in the following table:

Description	<u>e#</u>	project	tool
Hypothesis	E001	DBMS	Python
	E001	Ecommerce	Virtualization
Conclusion	E001	DBMS	Virtualization
	E001	Ecommerce	Python

However, the actual use of template dependency may be to represent the constraints that cannot be represented using FD, MVD and join dependency. The following example explains this.

Example: Consider the relation EMP (empID, empName, salary, headID). In this relation the attribute headID represents empID of the head of the employee. You want to put a constraint in the relation that the employee cannot be given more salary than his/her head. The following template dependency would define this constraint:

Description	<u>empID</u>	empName	salary	headID
Hypothesis	E1	N1	S1	E2
	E2	N2	S2	E3
Conclusion	$S2 \geq S1$			

The Domain-Key Normal Form (DKNF)

The Domain-Key Normal Form (DKNF) is beyond 5NF and proposes to make a set of relations free of all anomalies. The DKNF was defined as a consequence of Fagin's theorem that states:

“A relation is in DKNF if every constraint on the relation is a logical consequence of the definitions of keys and domains.”

Let us define the key terms used in the definition above – *constraint*, *key* and *domain* in more detail. These terms were defined as follows:

Keys can be either the primary keys or the candidate key. Let R be a relation schema with $CK \subseteq R$. A key CK requires that CK be a super key for schema R such that $CK \rightarrow R$. Please note that a key declaration is a functional dependency but not all functional dependencies are key declarations.

Domain is the set of definitions of the contents of attributes and any limitations on the kind of data to be stored in the attribute. Let A be an attribute and dom be a set of values. The domain declaration can be stated as $A \sqsubseteq \text{dom}$. It requires that the values of A in all the tuples of R be values from the set dom .

Constraint is a well-defined rule that is to be upheld by any set of legal data of R . A *general constraint* is defined as a predicate on the set of all the relations of a given schema. The MVDs, JDs are the examples of general constraints. However, a general constraint need not be just functional, multivalued, or join dependency. For example, the first two digits of your enrolment number represent the year in which you have taken admission. Assuming that MCA is the only programme of the university, whose maximum duration is 4 years. Also, assume that admission to this programme was started in 2021. Therefore, in the year 2026, there would be two types of students, viz. students whose enrolment number starts with 21 and students whose registration number starts with 22, 23, 24, 25 and 26. The registration of the students, whose registration starts with 21 would become invalid. Therefore, the general constraint for such a case may be: “If the first two digits of $t[\text{enrolmentnumber}]$ is 21, then $t[\text{marks}]$ are valid.”

The t represents a tuple and *enrolmentnumber* and *marks* are the attributes.

The constraint suggests that the database design is not in DKNF. To convert this design to DKNF design, you need two schemas as:

Validstudentschema = (*enrolmentnumber*, *subject*, *marks*)

Invalidstudentschema = (*enrolmentnumber*, *subject*, *marks*)

Please note that the schema of valid students requires that the enrolment number of the students begin with 22. The resulting design is in DKNF.

Although DKNF is an aim of a database designer, it may not be implemented in a practical design.

☛ Check Your Progress 3

- 1) Define Inclusion Dependencies.

.....
.....
.....

2) Define the template dependency.

.....
.....

3) What is the key idea behind DKNF?

.....
.....
.....

6.6 SUMMARY

This unit explains the concept of multi-valued dependencies, which causes a relation to have data redundancy. This causes a relation to have data anomalies. MVD is a consequence of having a set of attributes in a relation that determines more than one multi-valued attribute, which are independent of each other. MVD forms the basis of decomposition of a relation into 4NF relations. Further, certain relations do not have any FDs and MVDs but have anomalies. Such relations, in general, consist of more than two independent attributes. These relations contain join dependency, that is the relation has several projections, which can be joined losslessly to produce original relation. The join dependency forms the basis for 5NF decomposition. The unit also discusses about the inclusion and template dependencies, which are designed to represent the constraints that cannot be assigned using FDs, MVDs and join dependencies. Finally, the unit introduces the concept of DKNF. You may refer to the further readings for more details on these topics.

6.7 SOLUTIONS/ANSWERS

Check Your Progress 1

- 1) An MVD is a constraint due to multi-valued attributes. A constraint is multi-valued if all the following conditions hold in a relation:
 - The relational must have at least three attributes out of which one should be multi-valued attribute.
 - One of the attributes can multi-determine the other two attributes.
 - The other two attributes, as stated above should be independent of each other.

- 2) There are two MVDs that exist in the relation. These are:

$\text{ENAME} \rightarrow\!\!\!\rightarrow \text{PNAME}$ and $\text{ENAME} \rightarrow\!\!\!\rightarrow \text{DNAME}$

Due to these two MVDs the relation suffers from insertion, update and deletion anomalies. This relation can be decomposed into the following projections, which are in 4NF.

EMP_PROJECT

ENAME	PNAME
Rohan	Big Data
Rohan	Machine Learning

EMP_DNAME

ENAME	DNAME
Rohan	AI Unit
Rohan	Analytics

- 3) The relational instance of SUPPLY relation shows that all three attributes are independent of each other as any supplier can supply any part to any project. Thus, there is no FD or MVD in the relation. Therefore, the relation is already in 4NF.

Check Your Progress 2

- 1) A join dependency is defined for a relation R and its projections, say R_1, R_2, \dots, R_n , as follows:
The join of relations R_1, R_2, \dots, R_n must be equal to the relation R .
- 2) A relation is said to be in 5NF if either it has trivial join dependencies, or every projection R_i of the relation R is a super key of R .
- 3) All the attributes of relation SUPPLY are independent of each other. Does the relation have join dependency (SNAME, PARTNAME), (PARTNAME, PROJNAME), (PROJNAME, SNAME)?
The three projections for the given instance of SUPPLY would be:

R1

SNAME	PARTNAME
XYZ Pvt Ltd	Bolt
XYZ Pvt Ltd	Nut
ABC Ltd	Bolt
Info Comm Ltd	Nut
ABC Ltd	Nail

R2

PARTNAME	PROJNAME
Bolt	Big Data

Nut	Machine Learning
Bolt	Machine Learning
Nut	AI
Nail	Big Data

R3

SNAME	PROJNAME
XYZ Pvt Ltd	Big Data
XYZ Pvt Ltd	Machine Learning
ABC Ltd	Machine Learning
Info Comm Ltd	AI
ABC Ltd	Big Data

The join of the first two projections (R1 and R2) on PARTNAME would be:

JoinR1R2

SNAME	PARTNAME	PROJNAME
XYZ Pvt Ltd	Bolt	Big Data
XYZ Pvt Ltd	Bolt	Machine Learning
XYZ Pvt Ltd	Nut	Machine Learning
XYZ Pvt Ltd	Nut	AI
ABC Ltd	Bolt	Big Data
ABC Ltd	Bolt	Machine Learning
Info Comm Ltd	Nut	Machine Learning
Info Comm Ltd	Nut	AI
ABC Ltd	Nail	Big Data

The join of relations JoinR1R2 with R3 on SNAME, PROJNAME would be:

SNAME	PARTNAME	PROJNAME
XYZ Pvt Ltd	Bolt	Big Data
XYZ Pvt Ltd	Bolt	Machine Learning
XYZ Pvt Ltd	Nut	Machine Learning
ABC Ltd	Bolt	Big Data
ABC Ltd	Bolt	Machine Learning
Info Comm Ltd	Nut	AI
ABC Ltd	Nail	Big Data

Please observe that the joined relation is same as the instance of relation SUPPLY. Therefore, the join dependency (SNAME, PARTNAME), (PARTNAME, PROJNAME), (PROJNAME, SNAME) holds over the relation SUPPLY. To convert SUPPLY relation to 5NF, you may decompose it into the three projections R1, R2 and R3 as shown above. Please notice that each of the relation R1, R2 and R3 now have trivial join dependency, therefore, are in 5NF.

Check Your Progress 3

- 1) An inclusion dependency defines the referential constraint on two attributes. An inclusion dependency holds if a projection on an attribute of a relation, which may be a foreign key, is a proper subset of projection of another attribute of another relation, where it is the primary key.
- 2) Template dependency is a generic representation of various dependencies. It consists of two parts – the hypothesis and conclusion.
- 3) DKNF is the “ultimate normal form”. It defines the terms keys, domains and constraints.



UNIT 7 STRUCTURED QUERY LANGUAGE

- 7.0 Introduction
 - 7.1 Objectives
 - 7.2 Introduction to SQL
 - 7.3 Data Definition Language
 - 7.4 Data Manipulation Language
 - 7.4.1 Data insertion, Updating and Deletion
 - 7.4.2 Data Retrieval
 - 7.5 GROUP BY Clause and Aggregate functions
 - 7.6 Data Control Language
 - 7.7 Summary
 - 7.8 Solutions/Answers
-

7.0 INTRODUCTION

As discussed in the previous Units, a relational database system consists of relations, which are normalised to minimise redundancy. The normalisation process involves the concepts of functional dependency (FD), multi-valued dependency (MVD) and join dependency (JD). The normalisation results in the decomposition of tables into smaller but non-redundant relations. After designing the normalised database system, you would like to implement it by using software called relational database management system (RDBMS). RDBMSs were designed to manage relational data. Some of these RDBMSs are – Oracle, DB2, SQL Server, MySQL, etc. All these RDBMSs are designed and developed by different organisations and use different data storage formats. A structured query language (SQL) is one of the standards, which was created for the transfer of information from a RDBMS. The SQL consists of three basic languages, viz. Data Definition Language (DDL), which is used for defining the structure of the relations in the form of SQL tables and indexes; Data Manipulation Language, which is used for input, modification and deletion of information in SQL tables; and Data Control Language, which is used for controlling the access rights on the data of SQL tables and indexes. This unit introduces these three languages and the basic set of commands used in SQL.

You must learn SQL to use database technologies, therefore, you are advised to go through this unit very carefully. You must practice the concepts learnt in this unit on a commercial RDBMS.

7.1 OBJECTIVES

After going through this unit, you should be able to:

- create SQL objects (tables, indexes, etc.) from a database schema;
 - insert data into database tables using SQL commands;
 - retrieve data from a database using SQL queries;
 - use the Group By and Having clauses of SQL;
 - Using aggregate functions of SQL;
 - Create access rights on different database objects using SQL.
-

7.2 INTRODUCTION TO SQL

The Structured Query Language (SQL) was initially designed at IBM by D D Chamberlin and R F Boyce for the relational model that was proposed in 1970 by E F Codd. SQL was designed as a fourth-generation programming language to produce multiple record output as a result of a single command. In addition, SQL commands did not require the specification of indexes or other information for data retrieval. Later, SQL was standardised by the American National Standard Institute (ANSI) and a number of standards of SQL have been created, starting from SQL:86, which was proposed in 1986, SQL:92, SQL:99 and SQL:2003. Several modifications have been proposed in SQL:2003 in the years 2006-2019. One of the key advantages of using SQL is that it is *the query language* of most RDBMSs. Thus, it facilitates the migration of a database from one RDBMS to another RDBMS. However, some differences can be found in the SQL implementation on several RDBMSs. This unit covers some of the basic features of SQL only.

Why did SQL become popular? One of the major reasons for SQL's popularity is that it allows you to write queries without specifying how the query would be executed. For example, in case you want to JOIN three relations or tables, say A, B, C, then SQL allows you to join these tables without specifying the sequence of joining. The decision about how to execute the joins, e.g. (A JOIN B) JOIN C or A JOIN (B JOIN C), what indexes and views are to be used, etc.; are left to the query parser, translator, and query optimiser of RDBMS. In addition, the syntax of SQL is closer to the English language. Thus, SQL queries are simpler to write and run. The following are the features of SQL in the context of RDBMS:

- SQL is non-procedural, as in SQL you just need to say what information is required by you and NOT how that information is to be acquired from the database.
- The syntax of SQL is closer to the English Language; thus, it is very easy to comprehend.
- In general, the output of a SQL query may be a single record or a group of records.
- An interesting feature of SQL is that it can be used at different levels of ANSI's three-level architecture of database system.

SQL includes the following three sub-languages:

- Data Definition Language (DDL): The basic focus of DDL is the commands that can be used to convert database schema into SQL tables, indexes, etc., or change of structure of the tables. These commands also allow you to define the constraints on the attributes of a table. The DDL commands are explained in the next section.
- Data manipulation language (DML): The purpose of data manipulation is twofold. First, it has commands to input, modify and delete the data in the SQL tables. Second, it has the SELECT command, which helps in the retrieval of data from one or multiple tables.
- Data control language (DCL): The purpose of these commands is to specify user's privileges to database users. These commands are very important in client/server databases with different types of users.

7.3 DATA DEFINITION LANGUAGE

The design of a database consists of the physical, conceptual, and external schema. To implement this design, you need to define these schemas using SQL. In this unit, we use commands to define the conceptual schema. The external schema-related commands are discussed in the next unit. As far as the physical schema is concerned, we will cover only a few commands. The conceptual schema, which

includes the relational schema and attributes, is implemented in RDBMS as a set of tables and columns, whereas the tuples of the relations are implemented with the help of rows or data records.

We will explain some of the basic DDL commands in this section with the help of an example. Consider two relations of a UNIVERSITY database system, namely STUDENT and PROGRAMME.

The student relation (STUDENT) has the following data: student ID (STID), which is also the primary key, the name of the student (STNAME), the programme code in which that student is registered (PROGCODE) and the mobile phone number of the student (STMOBILE). Please note that this schema assumes that a student is allowed to register in only a single programme for which s/he is allotted a unique student ID.

The second relation, PROGRAMME, has the following data: programme code (PROGCODE), which is also the primary key, the name of the programme (PROGNAME) and the fee for that programme.

STUDENT (STID, STNAME, PROGCODE, STMOBILE) and
PROGRAMME (PROGCODE, PROGNAME, FEE)

The first step would be to define the datatypes of various attributes. We propose to use the following data types for the attributes.

	Relation Name	STUDENT			
Attribute	<u>STID</u>	STNAME	PROGCODE	STMOBILE	
Data type	Character	Character	Character	Number	
Length	4	40	6	12 digits	
Constraint	PRIMARY KEY	NOT NULL	FOREIGN KEY References PROGRAMME table	-	

	Relation Name	PROGRAMME	
Attribute	<u>PROGCODE</u>	PROGNAME	FEE
Data type	Character	Character	DECIMAL
Length	6	30	5
Constraint	PRIMARY KEY	NOT NULL	>1000 and < 50000

Figure 1: Example Relations

Once you have completed the data design, the next step would be to use SQL to create the SQL tables. To do so, you should learn about different data types in SQL, commands for creating tables in SQL, commands for creating constraints, etc. Let us discuss each of these.

Data Types in SQL: SQL supports many data types. Figure 2 lists some of the commonly used data types of SQL. For more data types supported by a DBMS, you may refer to the information manual of that DBMS. You may select one of these data types for each column.

CHAR (<i>n</i>)	It accepts a character string of size <i>n</i> . The character string can include alphabets, numeric digits, and special characters. The size of each string is fixed, that is, <i>n</i> .
-------------------	--

VARCHAR (<i>n</i>)	It accepts a character string up to the size <i>n</i> . The character string can include alphabets, numeric digits, and special characters.
BOOLEAN	It accepts a value False (zero value) or True (non-zero value)
INTEGER (<i>n</i>)	It can store signed integers or unsigned integers of length 32 bits. <i>n</i> represents the display width of the integer.
DECIMAL (<i>n, d</i>)	It can store decimal numbers of size <i>n</i> having <i>d</i> digits after the decimal point. The default value of <i>d</i> is 0.
DATE	Represents a valid date. It uses 4 digits for years and 2 digits each for month and day.
TIMESTAMP	It assigns a timestamp, which can be used to determine the recency of data.

Figure 2: Some Basic Data Types

For the relations of Figure 1, you may select CHAR for STID, PROGCODE, STMOBILE (as it is not required for computation), and VARCHAR for STNAME and PROGNAME, as the names of students may be of different lengths; so are the names of the programme's. The data type of FEE is DECIMAL(5).

Creating the Database and the Tables: In order to create the tables as given in Figure 1, you need to first create a database using the following command:

```
CREATE DATABASE <name of the database>;
```

```
USE <name of the database>; //This command is needed when the DBMS has multiple databases.
```

The following commands will create the UNIVERSITY database:

```
CREATE DATABASE UNIVERSITY;
```

```
USE UNIVERSITY;
```

Now, you can create the tables using the create table command. The following is the syntax of this command:

```
CREATE TABLE <name of the table> (
    ColumnName1 <data type> [constraints],
    ColumnName2 <data type> [constraints],
    ...
);
```

The following are the descriptions of the create table command:

- The name of a table should begin with an alphabet. It should not contain blank space and special characters except the underscore character (_).
- You should not use reserved words as a name of a table.
- You should use a unique name for each column in a table.
- The constraints are optional and therefore, shown in [] brackets.
- The data type should include the size of the column.
- You may use any of the following constraints on the column:

NOT NULL	This column of a table cannot be left blank while data entry or modification.
UNIQUE	This value should be unique across all the values in this column.
PRIMARY KEY	The column is or is a part of the primary key.
CHECK	It is followed by certain conditions, which should be fulfilled by the column.
DEFAULT	When a default value is specified for a column.
REFERENCES	This is used for specifying referential constraints, while creating a table.

Figure 3: Some of the constraints in Create Table Command

Now, you are ready to create the tables. Since the table STUDENT contains a reference to PROGRAMME table, therefore, you may create the PROGRAMME table first using the following command:

```
CREATE TABLE PROGRAMME
(
    PROGCODE CHAR(6) PRIMARY KEY,
    PROGNAME VARCHAR(30) NOT NULL,
    FEE DECIMAL (5),
    CHECK (FEE>1000 AND FEE<50000)
);
```

Now, you are ready to create the STUDENT table. You will use the following command to create the table.

```
CREATE TABLE STUDENT
(
    STID CHAR(4) PRIMARY KEY,
    STNAME VARCHAR(40) NOT NULL,
    PROGCODE CHAR (6),
    STMOBILE CHAR (12),
    FOREIGN KEY PROGCODE REFERENCES PROGRAMME (PROGCODE)
    ON DELETE RESTRICT
);
```

The Referential Action: Please note that in the command of creating the STUDENT table, we have used a referential action RESTRICT, which will make sure that any deletion of a record in PROGRAMME table will be restricted in case even one record of STUDENT table contains that PROGCODE. Thus, this action will ensure that there is no violation of the referential integrity constraint during deletion of a record from the PROGRAMME table. The other possible referential action is CASCADE. In this case, the deletion of a record in PROGRAMME table will result in the deletion of all the records of all the students whose PROGCODE is the same as the record, which is getting deleted in the PROGRAMME table.

Creating an Index: You may notice that the primary key of the STUDENT table is STID, therefore, the STUDENT table would be organised in the order of STID. However, many database queries may require the student records in the order of PROGCODE. This would require you to create an index on PROGCODE in the STUDENT table to enhance the query performance. The following command can be used to create an index:

```
CREATE [UNIQUE] INDEX <name of the index>
    ON <name of the table> (ColumnName1, ColumnName2, ...)
```

You use the keyword UNIQUE, only if a unique index is to be created. For the given example, you may create the following index for the STUDENT table.

```
CREATE INDEX PROGINDEX ON STUDENT (PROGCODE);
```

Other DDL commands: There are a large number of DDL commands. We will discuss only a few commands here. You may refer to the DBMS documentation for more commands.

Commands to alter a Table: For altering a table, you may use an ALTER TABLE command. This command can be used for performing the following functions:

- You can add a new column to an existing table, or you can modify a column of an existing table, by using the command:
ALTER TABLE <name of the table> ADD/MODIFY (ColumnName1, <datatype>, ...);
- You can add a new constraint in a table using the following command:
ALTER TABLE <name of the table> ADD CONSTRAINT
<name of the constraint> <type of the constraint> (ColumnName);
- You can drop a constraint on a table or enable or disable it using the following command:
ALTER TABLE <name of the table> DROP/ENABLE/DISABLE <name of the constraint>;

Commands to delete a Table or an Index: You can remove a table or an index, which is not required any more. The following commands can be used for these purposes.

DROP TABLE<name of the table>;

DROP INDEX <name of the Index> ON <name of the table>;

Commands to create a Domain: As defined earlier, SQL has a large set of data types. However, in many database implementations, you need to create a more meaningful domain that can define the data types and constraints on the data of a specific attribute or column. The following command can be used to create domains. It may be noted that the following command may not be defined in many DBMSs.

CREATE DOMAIN <Name of the Domain> AS <Data type> CHECK (<Constraints>);

You may refer to the DBMS documentation for more details.

Check Your Progress 1

- 1) List the advantages and disadvantages of using SQL.

.....
.....
.....

- 2) Consider a hotel that has three tables: ROOM (RNo, RType, RRent,), BOOKING (CustID, RNo, BookedFrom, BookedTo) and CUSTOMER (CustID, CustName, CustAddress, CustPhone). Assume the structure of these tables and create the tables using SQL. You may assume the following constraints for the HOTEL database:

- The type of hotel rooms can be: Single Room and Double Room, default room type should be Single room.
- The room rent is between 5000 per day to 25000 per day.
- Rooms are on different floors and numbered from 101 to 150.
- While booking the room the BookedFrom and BookedTo should follow the following relationship: Today's Date <= BookedFrom <= BookedTo
- No room should be booked twice for a specific date.

7.4 DATA MANIPULATION LANGUAGE

Once you have created a database and database tables along with the necessary constraints, the next step is to insert data in the tables. While inserting the data in the tables, you may commit some mistakes or there may be the need of making certain changes in the data of the tables, therefore, you would be requiring SQL commands to INSERT, UPDATE and DELETE records in a database table. These are Data manipulation language (DML) commands. These DML commands allow you to input and edit the data in the tables. Further, you would like to retrieve selected information from the database. In this section, first, we discuss the command to insert, update or delete the data followed by commands to retrieve information from the database. You may please note that the changes made by the DML statements are made permanent only after these operations are COMMITTED. You will learn about COMMIT in Unit 9.

7.4.1 Data Insertion, Updating and Deletion

The DML commands are used for inputting and editing data in a database table. To insert data in a table, you may use the insert command, which is explained next.

Inserting Data: The following command is used to insert a record into a table. The following two formats of insert commands are used:

If you are inserting a record that had data for all the columns, then you can simply use the command format:

```
INSERT INTO <name of the table> VALUES (v1, v2, v3, ...);
```

Please note that v1, v2, etc. are the values that are to be inserted into the respective column of the database. For example, to insert data into the PROGRAMME table, you can use the following INSERT command:

```
INSERT INTO PROGRAMME  
VALUES ("PGDCA", "Postgraduate Diploma in Computer Applications", 22000);
```

Please note the following with respect to the insert command, given above:

- The values inserted into the table would be PROGCODE as *PGDCA*; PROGNAME as *Postgraduate Diploma in Computer Applications*; and FEE as *22000*.
- You can use parameters instead of actual values, for example, you can use `INSERT INTO PROGRAMME VALUES (&1, &2, &3)`; These parameter values can be input at the time of execution of the query.
- You can use a sub-query (which will be explained in the next unit) instead of the values given in the command in the (...).

However, if you do not want to insert values in all the columns of the table, then you need to use the following format:

```
INSERT INTO <name of the table> (C1, C2, C3, ..., Cn) VALUES (v1, v2, v3, ..., vn);
```

Here, C1, C2, ... represents the column names and v1, v2, ... represents the corresponding value of a column. Please note that you need to specify n values for the n columns. For example, suppose you do not know the mobile number of a new student, still his/her information can be entered in the STUDENT table using the following SQL command:

```
INSERT INTO STUDENT (STID, STNAME, PROGCODE)
    VALUES ("1001", "RAMESH SHARMA", "PGDCA");
```

Please note the following with respect to the insert command, given above:

- The insertion is possible as STMOBILE has no constraint. It can be left blank.
- The values inserted into the table would be STID as 1001; STNAME as RAMESH SHARMA; PROGCODE as PGDCA; and STMOBILE as NULL.
- Please also note that the above insert statement will not cause a violation of foreign key constraint, as we have already inserted the PGDCA programme details in the PROGRAMME table.

Updating Data: For updating data, you may use the update command of SQL. The format of the command is given below:

```
UPDATE <name of the table>
SET <C1> = <v1>
WHERE <conditional statement>;
```

For example, to update the mobile number data of student 1001 to the number, say 8484848484, you can use the command:

```
UPDATE STUDENT
SET STMOBILE = "8484848484"
WHERE STID = "1001";
```

You can also use a subquery in an update statement (subqueries are covered in unit 8). For example, consider the fee of all the programmes, which has more than 100 students, is raised by 5%, then the following update command may be used to update the PROGRAMME table.

```
UPDATE PROGRAMME
SET FEE = FEE * 1.05
WHERE PROGRAMME.PROGCODE IN (
    SELECT STUDENT.PROGCODE
    FROM STUDENT
    GROUP BY STUDENT.PROGCODE
    HAVING COUNT(*) > 100);
```

The purpose of this subquery will be clear after you go through the next subsection. The subquery will find those programmes which have more than 100 students.

Deleting Data: You can use the following command to delete one or more records from a table:

```
DELETE FROM <name of the table>
WHERE <conditional statement>;
```

The delete command may delete one or more data records at a time. For example, you can try deleting the data of the PGDCA programme from your database, using the following command.

```
DELETE FROM PROGRAMME
WHERE PROGCODE= "PGDCA";
```

However, as there exists a foreign key constraint with referential action RESTRICT and you have already inserted a student who has PGDCA as his programme, therefore, DBMS will not allow you to delete the programme PGDCA from the PROGRAMME table. In case, you want to do so you need to first delete records of all the students of PGDCA from the STUDENT table and then you can delete the PGDCA programme from the PROGRAMME table. Please note that you can use subqueries instead of conditional statement in deletion also.

7.4.2 Data Retrieval

One of the most popular features of any DBMS is the ad-hoc query facility, which requires data retrieval as per the need and access rights of the user. SELECT statement is one of the most used statements of DML, as it helps in the retrieval of requisite data. In this section, we discuss various clauses of this statement.

SELECT Statement: The following is the basic format of the select statement.

```
SELECT      <List of Column names or expressions to be displayed>
FROM        <List of Tables that contain the data, whose columns are in select>
WHERE       <Conditions for selection of records for display>;
```

The following are examples of the use of this statement for the retrieval of data from the two relations given in Figure 1.

Example 1: List the details of all the programmes of the University.

For answering this query, are using one wildcard character (*), which represents all the columns of a table. Please note that in case, you have used * in an arithmetic expression in the SELECT clause, it will be treated as a multiplication sign.

```
SELECT      *
FROM        PROGRAMME;
```

This statement will display the PROGCODE, PROGNAME and FEE of all the records in the PROGRAMME table.

Example 2: List the programme code and programme names of all the programmes of the university, whose fee is less than or equal to 5000.

```
SELECT      PROGCODE, PROGNAME
FROM        PROGRAMME
WHERE       FEE <= 5000 ;
```

Please note the select clause now contains only those column names, which are to be displayed.

Example 3: List the id, name and mobile number of all the PGDCA students.

```
SELECT      STID, STNAME, STMOBILE
FROM        STUDENT
WHERE       PROGCODE = "PGDCA" ;
```

Example 4: This example shows the use of the **arithmetic operator** and **alias**. What would be the fee of the PGDCA programme, if students are given a discount of 15%?

```
SELECT      PROGCODE, PROGNAME, FEE*0.85 AS DISCOUNTEDFEE
FROM        PROGRAMME
WHERE       PROGCODE = "PGDCA" ;
```

Please note the computation of the fee and assigning the result of the computation a new name, that is an alias, DISCOUNTEDFEE. Please note that the syntax of assigning an alias may be different in different RDBMS.

Even in SQL expressions, the operator precedence is followed. Further, these expressions are executed from the left towards the right.

Example 5: This example shows the use of the **concatenation operator**. If you want to display the names of the programme as: “PGDCA: Postgraduate Diploma in Computer Applications” programme, you should use the following SQL command.

```
SELECT      PROGCODE + ":" + PROGNAME  
FROM        PROGRAMME  
WHERE       PROGCODE = "PGDCA";
```

Please note that in this statement the concatenate operator is ‘+’. However, in different DBMSs it may be different, for example, MySQL uses CONCAT() function, whereas Oracle uses || as a concatenation operator. In addition, please also note the use of the literal character string “:” in the command. A literal string may not be part of any column value but can be added to enhance information display.

Example 6: This example demonstrates how duplicate rows can be eliminated using the **DISTINCT** operator. Find the programme code of those programmes, which has at least one student.

To find the answer to this query, you may use the STUDENT table and Project it on programme code.

```
SELECT      DISTINCT PROGCODE  
FROM        STUDENT
```

In case you do not use DISTINCT then you will get duplicate values of programme code.

Example 7: This example demonstrates the use of the range operator **BETWEEN ... AND**. To find the list of programmes whose fee is ≥ 5000 but ≤ 15000 , you may use the following command:

```
SELECT      *  
FROM        PROGRAMME  
WHERE       FEE BETWEEN 5000 AND 15000;
```

Please note that both the values, viz. 5000 and 15000 are included in the range.

Example 8: This example demonstrates the use of the set operator **IN**. Find the students of PGDCA or BCA programmes. One way of answering that query would be:

```
SELECT      *  
FROM        STUDENT  
WHERE       PROGCODE IN ("BCA", "PGDCA");
```

Example 9: This example demonstrates the use of **LIKE** operator for matching a pattern of characters in the columns which are of CHAR or VARCHAR type. It may be noted that LIKE operator is supported by several wildcard characters, which may be different in different DBMS. In this example, we demonstrate the use of % wildcard character that matches zero or many characters in a string. For example, a string like %COM% will match with strings: COMPUTER, COMMERCE, INCOME, MCOM etc.

To find the list of all the programmes, which have word “Computer”, you may give the command:

```
SELECT      *  
FROM        PROGRAMME  
WHERE       PROGNAME LIKE "%Computer%";
```

Example 10: This example demonstrates the use of **IS NULL** operator. Find the list of students, whose mobile number is not with the University.

```
SELECT      *
FROM        STUDENT
WHERE       STMOBILE IS NULL ;
```

Example 11: SQL uses the logical operators, i.e., NOT, AND and OR. The precedence of these operators is shown in the following table:

Operators in increasing order of Precedence
Comparison operators (e.g. <, =, > etc.)
NOT
AND
OR

To print the name of all the students of PGDCA or BCA can also be written as:

```
SELECT      STID, STNAME
FROM        STUDENT
WHERE       PROGCODE = "BCA" OR PROGCODE = "PGDCA" ;
```

Ordering of Results using ORDER BY clause

The SELECT statement of SQL, in addition to SELECT, FROM and WHERE clauses, supports three more clauses, viz. ORDER BY clause, GROUP BY clause and HAVING clause. We will discuss the ORDER BY clause here and the remaining two clauses are discussed in the next section.

The ORDER BY clause is used, when you want to display the records in the sorted sequence of one or more columns. This sorted order can be increasing or decreasing. Also, you can use more than one column for sorting the data. Please note the following points about ORDER BY clause:

- ORDER BY is the last clause of a SELECT statement.
- You can mention ASC for ascending order or DESC for descending order. In case you do not mention any order, then ordering, by default, is in ascending order.
- You can sort the records on a column, which is part of the table given in FROM statement, even if you have not displayed that column.
- You can also sort on the alias that has been created by you in the select statement.

Example 12: List the name of the students in the order of programme and within a programme in alphabetical order.

```
SELECT      PROGCODE, STNAME
FROM        STUDENT
ORDER BY    PROGCODE, STNAME ;
```

Check Your Progress 2

- 1) Write the SQL commands to insert the following data in the STUDENT and PROGRAMME table. Highlight the errors, if any.

Relation Name	PROGRAMME	
PROGCODE	PROGNAME	FEE
BCA	Bachelor of Computer Applications	48000
MCA	Master of Computer Applications	60000

Relation Name	STUDENT		
STID	STNAME	PROGCODE	STMOBILE
1002	Abc	BCA	
1003	NULL	PGDCA	9199999999
1004	XYZ	MCA	

- 2) List the various clauses of the SELECT statement giving their purpose.
-
-
-

- 3) Consider the following two relations – S and SP.

S

SupplierNo	SupplierName	SupplierRanking	SupplierCity
S1	ABC	60	Delhi
S2	DEF	30	Mumbai
S3	EFG	50	Bangaluru
S4	FGH	40	Chennai
S5	GHI	NULL	Delhi

SP

SupplierNo	PartNo	QuantitySold
S1	PA	1000
S2	PA	500
S2	PB	1200
S3	PB	700
S5	PA	2100
S5	PB	1200

- a) Find the name of the suppliers, whose ranking is lower than 40 and the city of the supplier is Chennai.
- b) List the names of the suppliers of Delhi city in the decreasing order of the supplier ranking.
- c) List the names of the supplier who have supplied the part PB (use IN operator).
- d) List the codes of all the suppliers who have supplied at least one part.

- e) List all the suppliers, who have “EF” in their names.
 - f) Get part numbers for parts whose quantity sold in a supply is greater than 1000 or are supplied by S2. (Hint: It is a retrieval using union).
 - g) List the names of the suppliers, whose ranking is not given.
-
.....
-

7.5 GROUP BY CLAUSE AND AGGREGATE FUNCTIONS

In the previous section, you have gone through the concept of data manipulation language (DML). We have discussed the SELECT statement and its various clauses. In a database system, several queries require DBMS to produce information about a group. For example, you may be interested in finding the average marks of the group of PGDCA students vis-à-vis BCA students. The SQL supports a GROUP BY clause for such cases. In addition, SQL also supports a number of functions that can find aggregate information for a group of records. These functions are required to find the sum, average, counting of records etc. These are called aggregate functions. The following table defines some of the important aggregate functions used in SQL.

count	Used to count the number of records
sum	Finds the sum of the data of a column
avg	Finds the average of the data of a column
max	Finds the maximum value from the data of a column
min	Find the minimum value from the data of a column

Figure 3: Some Aggregate functions in SQL

Let us explain the use of these functions with the help of a few examples.

Example 13: Find the number of students of PGDCA.

```
SELECT      COUNT(*)
FROM        STUDENT
WHERE       PROGCODE = "PGDCA";
```

Please note that the wildcard *, in this case, indicates all the records to be counted, which fulfil the WHERE condition.

Example 14: Find the minimum, maximum and average fees of all the programmes.

```
SELECT      MIN(FEE), MAX(FEE), AVG(FEE)
FROM        PROGRAMME
```

GROUP BY clause

GROUP BY clause can be used to group records on certain criteria, e.g., you can group students on the basis of their Programmes. This clause is added after WHERE clause in the SELECT statement. In a SELECT statement in which you have used a GROUP BY clause, you can only use the aggregate functions or the column name on which you have grouped the data in the SELECT clause. The following example demonstrates this aspect:

Example 15: Find the number of students in every programme of the University.

You can answer this query by grouping the records on PROGCODE and finding the count of the student ids in each group.

```
SELECT      PROGCODE, COUNT(STID)
FROM        STUDENT
GROUP BY    PROGCODE;
```

Please note that in the SELECT clause of the SELECT statement above, we have used only the PROGCODE and the aggregate function COUNT.

HAVING clause

The HAVING clause can be used to specify a condition for a group. It is different from the WHERE clause, which is applicable for all the records, whereas the condition specified in the HAVING clause is to be fulfilled by a group of data. The following example explains the use of the HAVING clause.

Example 15: Count the number of students in each programme, where the mobile number of students is not given. List only those programmes which have more than 5 such Students.

```
SELECT      PROGCODE, COUNT(STID)
FROM        STUDENT
WHERE       STMOBILE IS NULL
GROUP BY    PROGCODE
HAVING     COUNT(STID) < 5;
```

Please note that in the SELECT clause of the SELECT statement above, we have used only the PROGCODE and the aggregate function COUNT.

7.6 DATA CONTROL LANGUAGE

The purpose of data control language (DCL) is to create users and assign access rights to them. In general, these commands are executed by a database administrator. The following are some of the most used DCL commands.

Creating a new user: You can create a new user using the following command:

```
CREATE USER <username for database user> IDENTIFIED BY <Password for the user>
```

For example, you can create a new user with the username “PGDCA_Student” with the password “PGDCA123”

```
CREATE USER PGDCA_Student IDENTIFIED BY PGDCA123
```

Use of GRANT Command: GRANT is used to give different kinds of accesses to a database user. Block3 covers the basic aspects of the GRANT option. In general, SQL supports two kinds of access permissions:

- The permissions that are at the system level.
- The permissions at the level of an object, such as a table, record, column etc.

The system-level permissions are, in general, specific to the DBMS environment, therefore, you may refer to the system documentation for details on such permissions. In this section, we provide a basic introduction to object-level permissions with the help of examples.

To give permission to get information from a table, the following SQL command may be used.

```
GRANT SELECT ON STUDENT, PROGRAMME TO PGDCA_Student;
```

To give permission for inserting and updating a record in the STUDENT table, you may use the following SQL command:

```
GRANT INSERT, UPDATE ON STUDENT TO PGDCA_Student;
```

In case you want to GRANT the SELECT access rights to more than one user on STUDENT table, then you may use the following command:

```
GRANT SELECT ON STUDENT TO PGDCA_Student1, PGDCA_Student2;
```

Use of REVOKE command: The REVOKE command is used to remove the access permissions that were granted to a user.

For example, you can also revoke SELECT permission using the following command:

```
REVOKE SELECT ON STUDENT FROM PGDCA_Student;
```

The following command will revoke all permissions of a user on STUDENT table.

```
REVOKE ALL ON STUDENT FROM PGDCA_Student;
```

Use of DROP command: You can remove a user using the DROP command.

For example, you can drop PGDCA_Student using the following SQL command:

```
DROP USER PGDCA_Student;
```

Check Your Progress 3

Consider the supplier relations given in Question 3 of Check Your Progress 2.

1) Write the SQL commands for the following aggregate operations.

- Find the total supply of part PA.
- Count the number of suppliers who have made a supply.
- List the part number and the total quantity supplied for that part.

2) Write the SQL commands for the following queries:

- List the suppliers who have made more than one supply.
- List the suppliers who have made more than one supply, with each supply being more than 1000.
- Find the part number and maximum quantity supplied, for the parts for which the total quantity supplied for that part is more than 1000. You may order the result in ascending order of part numbers.

3) Write the SQL commands for the following:

- Create a new user ABC having the password XYZ.
- Grant the user ABC to read table S only.
- Cancel all the access permissions of ABC on table S.

7.7 SUMMARY

SQL is one of the most important languages for any RDBMS. It allows a user to create tables, insert and update data in the tables and retrieve information from the tables. In addition, data of a table can be made available to only authorised users. This unit first introduces you to basic aspects of SQL and thereafter discusses the DDL commands. It is important that while creating tables you also include the constraints that are to be fulfilled by the tables. The constraints in SQL may be implemented using CHECK clause, PRIMARY KEY clause, FOREIGN KEY clause etc. You must also implement the referential action in case of referential integrity. You can also alter the tables if needed. This unit discusses the DDL commands for all the above operations. After creating the table using the DDL commands, next you use the DML commands to insert data into the tables. In case of any changes in the data values in the table, you may use the UPDATE command of DML. The commands for data insertion and updating have been discussed in the unit. One of the most important DML commands is SELECT which allows the retrieval of data from the table based on various criteria. This unit discusses various clauses of SELECT statement, viz., SELECT, FROM, WHERE, GROUP BY, HAVING and ORDER BY clauses. Finally, the unit discusses the CREATE USER, GRANT, REVOKE and DROP commands of DCL. You may please note that this unit does not cover all the SQL commands. You must practice these commands and learn more SQL commands from the user documentation of DBMS that you may use.

7.8 SOLUTIONS/ANSWERS

Check Your Progress 1

- 1) The following are the advantages of using SQL:
 - It is a standard query language across RDBMSs; therefore, you can port your data from one DBMS to another. Also, you just need to master SQL to use any RDBMS.
 - SQL is more English-like, so is easy to learn and use.
 - SQL exists for both interactive environments and as embedded SQL. Thus, you can use SQL for ad-hoc queries as well as in programs, like web-based programs.
- Some of the disadvantages of SQL are:
 - A query can be implemented in many ways, this may confuse a user.
 - The language, in its present form, is very large.
 - Some of the functions of the SQL are not portable across DBMS.
 - The result of an SQL command may allow duplicates, which are not relational in nature.
- 2) In this implementation, a lot of constraints have been defined, so instead of directly putting them in tables, first, we define the domains and use these domains in the CREATE table command. This will be a neater and more maintainable implementation.

CREATE DOMAIN TYPEOFROOM AS CHAR (1) CHECK (VALUE IN (S, D));

//Please note this constraint may not work on certain DBMS, where NOT EXISTS clause is not supported.

Check Your Progress 2

- 1) You may use the following sequence of instructions:

INSERT INTO PROGRAMME

VALUES (“BCA”, “Bachelor of Computer Applications”, 48000);

The above insertion will be successful.

INSERT INTO PROGRAMME

VALUES (“MCA”, “Master of Computer Applications”, 60000);

The above insertion will fail, as the FEE is more than the allowable fee value.

INSERT INTO STUDENT (STID, STNAME, PROGCODE)

VALUES (“1002”, “Abc”, “PGDCA”);

The above insertion will be successful, as PGDCA was inserted in Section 7.4.1

INSERT INTO STUDENT (STID, PROGCODE, STMOBILE)

VALUES (“1003”, “BCA”, “9199999999”);

The above insertion will fail, as the Student name column value must be given, as it has a constraint NOT NULL.

INSERT INTO STUDENT (STID, STNAME, PROGCODE)

VALUES (“1003”, “XYZ”, “MCA”);

The above insertion will fail, as the MCA programme insertion had failed earlier, so it will result in a violation of the referential constraint.

- 2) The following are the basic clauses of SELECT statements.

SELECT is used to specify the columns or expressions that are to be displayed in the result.

FROM is used to list the tables that are to be used for data output.

WHERE is used to specify conditions for the selection of data for the display.

GROUP BY clause is used to specify the columns, whose values should be used to group the records of the table.

HAVING is used to specify conditions, which should be fulfilled by each group for selection for display.

ORDER BY is used to specify the ordering of the records for the output.

- 3) a) SELECT SupplierName

FROM S

WHERE SupplierRanking < 40 AND SupplierCity = “Chennai”;

The output of this will be:

SupplierName
FGH

- 3) b) SELECT SupplierName

FROM S

WHERE SupplierCity = “Delhi”

ORDER BY SupplierRanking DESC;
The output of this will be:

SupplierName
ABC
GHI

- c) SELECT SupplierName
 FROM S
 WHERE SupplierNo IN (SELECT DISTINCT SupplierNo
 FROM SP
 WHERE PartNo= "PB") ;

The output of this will be:

SupplierName
DEF
EFG
GHI

- d) SELECT DISTINCT SupplierID
 FROM SP;

The output of this will be:

SupplierNo
S1
S2
S3
S5

- e) SELECT *
 FROM S
 WHERE SupplierName LIKE %EF% ;

The output of this will be:

SupplierNo	SupplierName	SupplierRanking	SupplierCity
S2	DEF	30	Mumbai
S3	EFG	50	Bangaluru

- f) SELECT DISTINCT PartNo
 FROM SP X
 WHERE X.QuantitySold > 1000
 UNION
 (SELECT DISTINCT PartNo
 FROM SP Y
 WHERE Y.SupplierNO = "S2") ;

The output of this will be:

PartNo

PA
PB

g) SELECT SupplierName
 FROM S
 WHERE SupplierRanking IS NULL ;
 The output of this will be:

SupplierName
GHI

Check Your Progress 3

- 1)
- a) SELECT sum(QuantitySold)
 FROM SP
 WHERE PartNo = “PA” ;
 - b) SELECT count(DISTINCT SupplierNo)
 FROM SP;
 - c) SELECT PartNo, sum(QuantitySold)
 FROM SP
 GROUP BY PartNo
- 2)
- a) SELECT SupplierNo
 FROM SP
 GROUP BY SupplierNo
 HAVING Count(PartNo) > 1 ; // As each supply is of one part only.
 - b) SELECT SupplierNo
 FROM SP
 WHERE QuantitySold > 1000
 GROUP BY SupplierNo
 HAVING Count(PartNo) > 1 ; // As each supply is of one part only.
 - c) SELECT PartNo, Max(QuantitySold)
 FROM SP
 GROUP BY PartNo
 HAVING Sum(QuantitySold) > 1000
 ORDER BY PartNo;
- 3)
- a) CREATE USER ABC IDENTIFIED BY XYZ;
 - b) GRANT SELECT ON S TO ABC;
 - c) REVOKE ALL ON S FROM ABC;

UNIT 8 Structured Query Language (Part-II)

Structure

- 8.0 Introduction
- 8.1 Objectives
- 8.2 SQL Joins
 - 8.2.1 Equi-join
 - 8.2.2 Outer Join
 - 8.2.3 Self-Join
- 8.3 Nested Queries
 - 8.3.1 Subqueries
 - 8.3.2 Correlated Subqueries
- 8.4 Database Objects
 - 8.4.1 Views
 - 8.4.2 Sequences
 - 8.4.3 Indexes and Synonyms
- 8.5 Summary
- 8.6 Solutions/Answers

8.0 INTRODUCTION

In the previous unit, you have gone through the details of the basic set of commands of the Structured Query Language (SQL). A good database system consists of normalised relations, which involve the lossless decomposition of relations into smaller relations. These smaller relations have controlled redundancy and help in removing the redundancy-related problems of a database, which you have studied in the earlier units of this course. However, when you want to retrieve information from a database, you may be required to extract information from several smaller relations. Join is one of the most important operations, which can be used to provide information from multiple tables. This unit discusses different types of join operations that can be performed on database tables.

In many database queries, you may be required to use a sub-query that provides data for the main query. This unit discusses the subqueries and correlated subqueries. The unit also introduces you to the concept of views, sequences, indexes and synonyms database objects. You must practice these queries, to master them.

8.1 OBJECTIVES

After going through this unit, you should be able to:

- Define different kinds of joins that can be performed on tables;
- Write the SQL statements for Join commands;
- Use subqueries and correlated subqueries;
- Write nested subqueries;
- Define and write SQL commands to create views and indexes;
- Define and write SQL commands for creating sequences and synonyms.

8.2 SQL JOINS

The join is an important relational operation for queries that involve more than one relation. You have already gone through the relational join operation in an earlier unit, this unit explains the use of the join operation in SQL. The purpose of join operation is to combine the data from two different tables on specific attribute(s), which share the same domain. *The necessary condition for the meaningful join of two tables is that each of these tables should have at least one attribute, called the joining attribute, that should have the same data domain.* The join operation uses the joining condition to combine the data of the two tables. In this unit, we will use the tables and data given in Figure 1 for various examples of join operations.

Table Name: CLIENT			
<u>ClientID</u>	ClientName	ClientAddress	ClientPhone
C001	ABCD	79, MGRoad	9999999991
C002	BCDE	Raman Street	9999999992
C003	CDEF	Vindyachal apatments	9999999993

Table Name: ITEM			
<u>ItemID</u>	ItemName	ItemPrice	QuantityAvailable
I01	Pen	20	500
I02	Pencil	5	1000
I03	Paper Sheet	20	1000
I04	Sharpener	5	200

Table Name: ORDER	
<u>OrderID</u>	ClientID
O001	C001
O002	C002
O003	C001
O004	C003
O005	C002

Table Name: ORDERDETAILS		
<u>OrderID</u>	<u>ItemID</u>	QuantityPurchased
O001	I02	2
O001	I03	10
O001	I04	1
O002	I02	5
O003	I01	2
O003	I03	5
O004	I01	5
O004	I03	3
O005	I01	8

Figure 1: Sample Tables and Data

The tables above do not have a date of purchase, as these tables are just for demonstrating various commands. Please note that the primary key of each table has been underlined. Please also note that each OrderID is unique, and an order may contain several items. The foreign keys in the tables are:

- OrderID in ORDERDETAILS table references ORDER table
- ItemID in ORDERDETAILS table references ITEM table
- ClientID in ORDER table references CLIENT table

Now, let us answer the following questions about the join operation:

1. Can you join the CLIENT and the ITEM tables? No, as there is no common attribute in these two tables on which table can be joined. Please note that two tables can be joined only if they have at least one attribute each which has the same domain.
2. How will you find the name of items that have been ordered by a client, whose name is ABCD? ClientID is C001? In order to answer it first let us assume that your DBMS supports the clause

```
SELECT * INTO # <Name of a Temporary Table>
FROM ...;
```

Now, you can find the relevant information in the following way (a very cumbersome way):

- a) First, find the ClientID of the client whose name is “ABCD” from the CLIENT table using the command:

```
SELECT ClientNo INTO #TempCLIENT
FROM CLIENT
WHERE ClientName = "ABCD";
```

The output of this query would be a Temporary table named #TempCLIENT. As per the data of Figure 1, the only ClientID with the name ABCD is C001. Thus, the output of the command above would be:

Table Name: #TempCLIENT
ClientID
C001

- b) Next, you can find the orders made by ClientID C001 from the ORDER table using the command:

```
SELECT OrderID INTO #TempORDER
FROM ORDER
WHERE ClientID IN (           SELECT *
                      FROM #TempCLIENT);
```

The output of the command above would be:

Table Name: #TempORDER
OrderID
O001
O003

- c) Next, you will find the items that have been ordered in each of the OrderID O001 and O003 from the ORDERDETAILS table using the command:

```
SELECT DISTINCT ItemID INTO #TempORDERDETAILS
FROM ORDERDETAILS
WHERE OrderID IN (           SELECT *
                      FROM #TempORDER);
```

The output of the command above would be:

Table Name: #TempORDERDETAILS
ItemID
I02
I03
I04
I01

- d) Now, you can find the name of the ordered items by C001, from the ITEM table using the following command:

```
SELECT ItemID, ItemName
FROM ITEM
WHERE ItemID IN (
    SELECT *
    FROM #TempORDERDETAILS);
```

The output of the command above would be:

ItemID	ItemName
I02	Pencil
I03	Paper Sheet
I04	Sharpener
I01	Pen

Figure 2: The Output of the Command

Can this query be answered with a single query? You may require the JOIN command to solve this query.

```
SELECT DISTINCT i.ItemID, i.ItemName
FROM CLIENT c, ORDER o, ORDERDETAILS oo, ITEM i
WHERE c.ClientName = "ABCD" AND
c.ClientID = o.ClientID AND
o.OrderID = oo.OrderID AND
oo.ItemID = i.ItemID
```

Though the statement above looks very complicated, you can interpret it by the following statements:

- In the select statement, you have put the information that you want to output. The ItemName information is available in only ITEM table, whereas ItemID is available both in ITEMDETAILS and ITEM tables, thus, the alias i before the ItemID is necessary and informing the database server that ItemID data is to be obtained from the ITEM table (please note i is an alias to ITEM table.).
- In the FROM clause have used aliases for every table that is being used.
- The WHERE clause has the following conditions:
 - c.ClientName = "ABCD", which will identify the ClientID as C001.
 - c.ClientID = o.ClientID will join the CLIENT table (alias c) with Order table (alias o), and output O001 and O003.
 - o.OrderID = oo.OrderID will join the ORDER and ORDERDETAILS tables
 - oo.ItemID = i.ItemID will join the ORDERDETAILS and ITEM tables.

Thus, producing the final result, as shown in Figure 2.

This example may seem to be very complex, but you must once again go through the example after going through the remaining part of this unit. Now, let us now focus on different kinds of joins.

8.2.1 Equi-join

Equi-join, as the name suggests, has equality as the joining condition. This means that the attributes that you are using to join would be checked for equality. Please note that the names of the joining columns in the two tables may be different. The following is an example of an equijoin operation:

Example 1: Using the tables of Figure 1 answer the query:

List the ID, name of the client, and the order ids of all the clients who have placed an order.

You need to use the CLIENT and ORDER tables to answer this query, as the name of the clients are in the CLIENT table and the order id information is in the ORDER table. You may use equi-join operation for this query using the columns ClientNo in CLIENT and ClientNo in the ORDER table. Though in this present example, both the columns have the same name, i.e. ClientID, equi-join operation does not require these names to be the same. SQL query for the query is presented below:

Query Using equi-join:

```
SELECT c.ClientID, c.ClientName, o.ClientID, o.OrderID
FROM CLIENT c, ORDER o
WHERE c.ClientID = o.ClientID;
```

The result of the query, on the tables of Figure 1, will be:

c.ClientID	ClientName	o.ClientID	OrderID
C001	ABCD	C001	O001
C001	ABCD	C001	O003
C002	BCDE	C002	O002
C002	BCDE	C002	O005
C003	CDEF	C003	O004

The output shows the ClientID of both the CLIENT and the ORDER tables. We can display any one of these two columns. Please also note that joining may result in the display of redundant information, as you can observe in the Client name column.

Natural Join: Natural join is one of the most used join operations. It requires that the joining columns of the two tables should have the same name. It uses the equality condition. The result of the join operation displays only one of the joining attributes/columns.

The query of example 1 can also be answered using the natural join operation, as both the tables have the same column name ClientID. The following SQL command will perform the natural join operation:

```
SELECT CLIENT.ClientID, ClientName, OrderID
FROM CLIENT NATURAL JOIN ORDER;
```

The output of the command would be:

CLIENT.ClientID	ClientName	OrderID
C001	ABCD	O001
C001	ABCD	O003
C002	BCDE	O002
C002	BCDE	O005
C003	CDEF	O004

Inner Join: Inner join combines two tables using a combining condition on the joining attributes. In the present versions of SQL, you may have to use INNER JOIN command for example 1 query:

```
SELECT CLIENT.ClientID, ClientName, OrderID
FROM CLIENT INNER JOIN ORDER
ON CLIENT.ClientID = ORDER.ClientID;
```

8.2.2 Outer Join

Consider the situation that a new client has been added to the client table and the present state of the CLIENT table is:

Table Name: CLIENT			
ClientID	ClientName	ClientAddress	ClientPhone
C001	ABCD	79, MGRoad	9999999991
C002	BCDE	Raman Street	9999999992
C003	CDEF	Vindyachal apartments	9999999993
C004	DEFG	Maidan Road	9999999994

Further, assume that this new client has not issued any order. Let us now issue the query of example 1 again on the present state of the database. You will find the result of the query will still be the same, as shown in example 1. So, we get no information about C004, in the result of the query.

If the objective of the query was to show the list of all the clients and their OrderIDs, irrespective of the fact, that they have given zero or more orders, then how would you extract such information? In effect, you want that all the records from the CLIENT table should participate in joining even if there is no joining row in the ORDER table. This requires the use of OUTER JOIN operation.

Example 2: List all the clients and their orders (NULL in case no order is given by a client).

The following SQL command will be required for the query asked above:

```
SELECT CLIENT.ClientID, ClientName, OrderID  
FROM CLIENT LEFT OUTER JOIN ORDER  
ON CLIENT.ClientID = ORDER.ClientID;
```

We have used the term LEFT OUTER JOIN, as in this command we want that all the records of the table mentioned on the left side of the join command (CLIENT in this case) be part of the output, irrespective of a joining record on the table on the right side (ORDER in this case). The output of the command would be:

CLIENT.ClientID	ClientName	OrderID
C001	ABCD	O001
C001	ABCD	O003
C002	BCDE	O002
C002	BCDE	O005
C003	CDEF	O004
C004	DEFG	NULL

You may notice that the last record in this table is for Client C004, who has not placed any order yet. This record is displayed as we have used the Left Outer join operation. Similarly, you can use the RIGHT OUTER JOIN or FULL OUTER JOIN, as per the need of database output.

8.2.3 Self-Join

In the self-join operation, a table is joined with a copy of itself. This is very useful for queries, which draw information from within a column of a table. The following is an example of the self-join.

Example 3: Find the pairs of similar priced items in the ITEM table.

You may first simply inspect the ITEM table. You will find that pairs -Pen and Paper Sheet; and Pencil and Sharpener, have the same price. How did you find this information? You compared the ItemPrice of each item with other ItemPrice. This inspection, in general, can be performed by join operation. Hence, you can answer the query using the following SQL command:

```
SELECT i1.ItemName, i2.ItemName
```

```

FROM ITEM i1, ITEM i2
WHERE i1.ItemPrice = i2.ItemPrice;

```

Thus, you are joining the two tables on identical ItemPrice, and displaying the pair of names of the items that have similar prices. However, you will get the following output of this SQL command:

i1.ItemName	i2.ItemName
Pen	Pen
Pen	Paper Sheet
Pencil	Pencil
Pencil	Sharpener
Paper Sheet	Paper Sheet
Paper Sheet	Pen
Sharpener	Sharpener
Sharpener	Pencil

You may observe that the output of the command has many extra records like records with the same item, e.g. the pair (Pen, Pen). In addition, a pair like (Pen, Paper Sheet) is same as (Paper Sheet, Pen), therefore, only one of these pairs should appear in the result. A simple solution to this problem is to add another condition in the SQL command: (i1.ItemID < i2.ItemID). This will ensure that only those pairs in which ItemID of first table is less than ItemID of second table will be part of output, thus, eliminating both the problems, as stated above. The command is as follows:

```

SELECT i1.ItemName, i2.ItemName
FROM ITEM i1, ITEM i2
WHERE i1.ItemPrice = i2.ItemPrice AND i1.ItemID < i2.ItemID;

```

Thus, you are joining the two tables on identical ItemPrice, and displaying the pair of names of the items that have similar prices. You will get the following output of this SQL command:

i1.ItemName	i2.ItemName
Pen	Paper Sheet
Pencil	Sharpener

Check Your Progress 1

Consider the Tables, given in Figure 1, and answer the following questions:

- 1) (a) Find the list of those item names that have been supplied, as part of at least one Order.
 (b) List the order details including the order id, item code, item name and price.
 (c) Compute the total price of an order.
-

- 2) Find the list of all the item IDs, item names and related Order IDs. This query should list the name of the items even if it is not part of any order.
-

- 3) Find the list of items that have been purchased together.

.....
.....
.....

8.3 NESTED QUERIES

In the previous section, we discussed join queries, which are responsible for joining the data from two tables. Nested queries are used when the output of a query may be useful for the execution of another query. Thus, you can create a main query nest a sub-query in any of the clauses of this main query. In this section, we discuss the basic type of nested queries and then a special type of nested subqueries called correlated subqueries.

8.3.1 Sub-queries

A sub-query is another SELECT statement that is used in the main SELECT statement. Please remember the following points about a subquery:

- A sub-query is executed prior to the main query. Therefore, you can use the result of a sub-query in an expression of the main query.
- A sub-query, on its execution, can return either a single value or a set of values or a relation. Therefore, the result of the sub-query can be used in a comparison in the WHERE or HAVING clause or for comparison with a set operator; or even in the FROM clause when a relation is returned.
- You can put a sub-query inside a sub-query. You can use a separate table in the main and sub-query.
- You should not use the ORDER BY clause in a sub-query, rather it should be used as the last clause of the main query. However, you can use the GROUP BY clause in the sub-query.
- When you use sub-queries in the WHERE or HAVING clause of the main query, you may be required to use comparison or set operators. Some of these operators are given below:

Comparison Operators	<, =, >=, <, <=, <>	Used when the sub-query returns a single constant value
Set Operators	IN, ANY, ALL	Used when sub-query returns a set of values.

These operators will be explained in various examples. We will use tables and data from Figure 1 to answer the following queries.

Example 4: Find the item, whose available quantity is the smallest.

```
SELECT      *
FROM        ITEM
WHERE       QuantityAvailable = ( SELECT      MIN (QuantityAvailable)
                                  FROM        ITEM);
```

The sub-query in this case will find a single minimum value, which will be compared in WHERE clause to determine the record, which has that minimum value.

ItemID	ItemName	ItemPrice	QuantityAvailable
I04	Sharpener	5	200

Example 5: Find the item, whose price is the minimum.

```
SELECT *  
FROM ITEM  
WHERE ItemPrice = ( SELECT MIN (ItemPrice)  
                      FROM ITEM);
```

The sub-query in this case has found a single minimum value, which will be compared in the WHERE clause. Please note that in this case, the output contains two records.

ItemID	ItemName	ItemPrice	QuantityAvailable
I02	Pencil	5	1000
I04	Sharpener	5	200

Example 6: Find the items, whose price is greater than the price of a Pencil and the quantity available is more than the Item whose ItemID is I01. Assume that item names are unique.

```
SELECT *  
FROM ITEM  
WHERE ItemPrice > ( SELECT ItemPrice  
                      FROM ITEM  
                     WHERE ItemName = "Pencil")  
  
AND  
QuantityAvailable > ( SELECT QuantityAvailable  
                           FROM ITEM  
                          WHERE ItemID = "I01");
```

There are two sub-queries in this case, the output of the first sub-query would be value 5 and the output of the second sub-query would be value 500. The result of the query is given below:

ItemID	ItemName	ItemPrice	QuantityAvailable
I03	Paper Sheet	20	1000

Example 7 (a): Find the total quantity of all the items in every order.

This query can be answered by the ORDERDETAILS table using a simple GROUP BY clause:

```
SELECT      OrderID, SUM(QuantityPurchased) AS TotalofAllItems  
FROM        ORDERDETAILS  
GROUP BY    OrderID;
```

OrderID	TotalofAllItems
O001	13
O002	5
O003	7
O004	8
O005	8

Example 7 (b): Find the total quantity of all the items for the orders, which have ordered more total quantity than ordered in order Q003.

Please note that the sub-query in this case is in the HAVING clause. This command will output:

OrderID	TotalofAllItems
O001	13
O004	8
O005	8

Example 8(a): Find the total price of each Order.

This query would require joining of ORDERDETAILS table and ITEM table, as the details of an order are in the ORDERDETAILS and Price of each order is in ITEM table. The following command would simply compute the Price of each Item of the order:

```
SELECT ORDERDETAILS.*, ItemName, ItemPrice, (QuantityPurchased*Price) AS ItemTotal  
FROM ORDERDETAILS, ITEM  
WHERE ORDERDETAILS.ItemID = ITEM.ItemID;
```

This command will list the orders as under:

OrderID	ORDERDETAILS.ItemID	QuantityPurchased	ItemName	ItemPrice	ItemTotal
O001	I02	2	Pencil	5	10
O001	I03	10	Paper Sheet	20	200
O001	I04	1	Sharpener	5	5
O002	I02	5	Pencil	5	25
O003	I01	2	Pen	20	40
O003	I03	5	Paper Sheet	20	100
O004	I01	5	Pen	20	100
O004	I03	3	Paper Sheet	20	60
O005	I01	8	Pen	20	160

To find the answer of the stated query, you need to find the sum of item total for each order. Thus, you can answer of the stated query by the following SQL command:

```
SELECT OrderID, SUM(QuantityPurchased*Price) AS OrderTotal  
FROM ORDERDETAILS, ITEM  
WHERE ORDERDETAILS.ItemID = ITEM.ItemID  
GROUP BY OrderID;
```

This command will list the orders as under:

OrderID	OrderTotal
O001	215
O002	25
O003	140
O004	160
O005	160

Example 8(b): Find the total price of the orders, which have at least one order for the Item whose item name is Paper Sheet.

This query would require you to use the answer of the query 8(a) and use of join in the subquery. The answer to the subquery would be a set of records, therefore, we will use set operator IN. The query is given as follows:

```

SELECT OrderID, SUM(QuantityPurchased*Price) AS OrderTotal
FROM ORDERDETAILS, ITEM
WHERE ORDERDETAILS.ItemID = ITEM.ItemID AND OrderID IN
    ( SELECT DISTINCT OrderID
      FROM ORDERDETAILS oo, ITEM i
      WHERE oo.ItemID = i.ItemID AND
            ItemName = "Paper Sheet")
GROUP BY OrderID;

```

This command will list the orders as under:

OrderID	OrderTotal
O001	215
O003	140
O004	160

Example 9: Find the id and name of all the clients, who have ordered Pen and Paper Sheet in a single order. This query would require the joining of the ORDER table and CLIENT table in the main query and the ORDERDETAILS table and ITEM table in the sub-query. The following command would find the list of OrderIDs, which have both Pen and Paper Sheet in a single Order.

```

(SELECT DISTINCT OrderID
  FROM ORDERDETAILS, ITEM
  WHERE ORDERDETAILS.ItemID = ITEM.ItemID AND ItemName= "Pen")
INTERSECT
( SELECT DISTINCT OrderID
  FROM ORDERDETAILS, ITEM
  WHERE ORDERDETAILS.ItemID = ITEM.ItemID AND ItemName= "Paper Sheet");

```

This query will result in the following output:

OrderID
O003
O004

Now, you can write the query as follows:

```

SELECT DISTINCT (CLIENT.ClientID, ClientName)
  FROM ORDER, CLIENT
 WHERE ORDER.ClientID = CLIENT.ClientID AND
       OrderID IN (
           (SELECT DISTINCT OrderID
             FROM ORDERDETAILS, ITEM
             WHERE ORDERDETAILS.ItemID = ITEM.ItemID AND ItemName= "Pen")
INTERSECT
           ( SELECT DISTINCT OrderID
             FROM ORDERDETAILS, ITEM
             WHERE ORDERDETAILS.ItemID = ITEM.ItemID AND ItemName= "Paper Sheet"))
);

```

This command will list the client's information as under:

ClientID	ClientName
C001	ABCD
C003	CDEF

Example 10: List the total price of the orders, whose total price is more than 150.

An alternate way of writing this query would be to use a subquery in the FROM clause, as given below:

```
SELECT OrderID, OrderTotal  
FROM (SELECT OrderID, SUM(QuantityPurchased*Price) AS OrderTotal  
      FROM ORDERDETAILS, ITEM  
      WHERE ORDERDETAILS.ItemID = ITEM.ItemID  
      GROUP BY OrderID)  
WHERE OrderTotal > 150;
```

OrderID	OrderTotal
O001	215
O004	160

You may please note that an alias in the subquery is used in various clauses of the main query.

8.3.2 Correlated Subqueries

In many queries, the columns of the tables used in FROM clause are also used in the subquery. Such subqueries are called Correlated subqueries and generally are time-consuming queries. These queries are explained with the help of the following example.

Example 11: List the client id and name of the clients, who have ordered ItemID I01 and ItemID I03, as part of a single order.

This query can be answered as a correlated query as follows:

```
SELECT DISTINCT OrderID  
FROM ORDERDETAILS outer  
WHERE ORDERDETAILS.ItemID = "I01" AND EXISTS  
      ( SELECT DISTINCT OrderID  
        FROM ORDERDETAILS inner  
        WHERE outer.OrderID=inner.OrderID AND  
              outer.ItemID<inner.ItemID AND inner.ItemID = "I03"  
      );
```

The execution of such a correlated query is time-consuming. Why? Since in these queries, the subquery is executed for each instance of the main query. For example, in the case of example 11, the main query will find that the order O003 fulfils the main clause. This will trigger the execution of the subquery, which will check that for the same value of order O003, there exists a record for item I03. Since it exists, therefore O003 will be output. Similarly, O004 will be output. In the case of O005, the main query has the record for I01, however, the record for I03 does not exist. Thus, O005 will not be output. Thus, this query will result in the following output:

OrderID
O003
O004

Check Your Progress 2

- 1) What is a subquery? When would you like to use the sub-query?

.....
.....
.....

- 2) Consider the following relations:

EMPLOYEE (EmployeeNo, EmployeeName, EmployeePhone, EmployeeBasicPay)

PROJECT(ProjectNo, ProjectName, ProjectDuration)

EMPLOYEEPROJECT (EmployeeNo, ProjectNo, EmpRoleInProject)

Now answer the following queries:

- (a) Find the name of the projects, which have the least duration.
(b) Find the name and basic pay of employees who are working on more than one project.
(c) Find the name of the employee who earns more than the average salary of all the employees.

.....
.....
.....

- 3) Consider the following relations Schema given in question 2 above, and the following two SQL queries on this schema. What is the purpose of these two queries?

(i) SELECT EmployeeName
 FROM EMPLOYEE
 WHERE EmployeeNo IN (SELECT EmployeeNo
 FROM EMPLOYEEPROJECT
 GROUP BY EmployeeNo
 HAVING COUNT(EmployeeNo) =
 (SELECT COUNT (*) FROM PROJECT));

(ii) SELECT ProjectName
 FROM PROJECT
 WHERE ProjectNo IN ((SELECT ProjectNo
 FROM PROJECT)
 MINUS
 (SELECT DISTINCT ProjectNo
 FROM EMPLOYEEPROJECT)
));

8.4 DATABASE OBJECTS

Database objects are useful concepts in a database system. In this section, we discuss four different types of objects, which are defined in many database management systems. Views are virtual tables, which may be used for implementing database security. In addition, they can also be used for database query optimisation. Sequences are used to maintain an automatic sequence of numbers, which can be very useful for input of unique values in a column. Indexes are used to enhance the performance of a database system. The following sub-section discusses these concepts in detail.

8.4.1 Views

A view is a part of external schema of a database, which can be used to restrict the data display, input and modification of a database system. A database system consists of base tables, which are the tables created for an entity or relationship of a logical or conceptual database model. The base tables are used to store data. A view can be categorised as a virtual table, which can be stored as a query on a table. This query is called the view definition. Please note that views do not store data of their own, rather data is stored in the base tables of the database. The query definition of a view can be stored in the data dictionary of DBMS. When you use a view, then it can be executed through the following steps:

- Retrieve the view definition from the data dictionary.
- Find the access privileges of the view, as well as the base tables associated with the view.
- Create the view qualified query, if the user has access privileges on the view and data of base tables.

The following are the Advantages of using views in a database system:

- Views allow database users to access only that data to which they have access privileges.
- Views mechanisms enforce logical data independence in a database system.
- Views allow different users to view the data differently.

How to create a view?

A view is created using a CREATE VIEW statement followed by a query. A view statement can include multiple tables in the FROM clause, the condition of joining two tables in the WHERE clause, the GROUP BY clause and expressions of a subquery. However, a view statement is not allowed to include an ORDER BY clause. The following are examples of view creation.

Example 12: Create a view for the logistics person, who would be interested in the list of all the items (with the item name and quantity) of a particular order, say O001, and the address of the client to whom that order is to be sent.

CREATE VIEW LOGISTICS AS

```

SELECT oo.OrderID OOID, oo.ItemID IItemID, ItemName, QunatityPurchased,
c.ClientID CClientID, ClientName, ClientAddress, ClientPhone
FROM ORDERDETAILS oo, ITEM i, ORDER o, Client c
WHERE oo.ItemID = i.ItemID AND oo.OrderID = o.OrderID
AND o.ClientID=c.ClientID AND oo.OrderID= "O001";

```

This will create a view named LOGISTICS. you can display the content of the LOGISTICS view using the following command:

```

SELECT *
FROM LOGISTICS;

```

It will display the following output:

OOID	IItemID	ItemName	QuantityPurchased	CClientID	ClientName	ClientAddress	ClientPhone
O001	I02	Pencil	2	C001	ABCD	79, MGRoad	9999999991
O001	I03	Paper Sheet	10	C001	ABCD	79, MGRoad	9999999991
O001	I04	Sharpener	1	C001	ABCD	79, MGRoad	9999999991

Data Updates through Views?

Views can be used for data updates though there are several restrictions on updating data through the views. In general, you cannot update the data through a view, if it contains a JOIN operation or a GROUP BY clause or an aggregate function or a subquery. In addition, you may use a check option for views, which is defined next.

Creating views with check option: Consider a view that allows updates on data, but those updates may result in a modified record, which does not fulfil the view defining criteria. The following example explains this situation.

Example 13: Create a view of the items whose price is more than INR 15.

```

CREATE VIEW COSTLYITEM AS
SELECT *
FROM ITEM
WHERE ItemPrice > 15;

```

This will create the view consisting of two items I01 and I03. Assume that the price of the Pen is discounted and made Rs 10, if you allow an update through the COSTLYITEM view, then the price of item I01 (Pen) will be updated to INR 10. Thus, I01 will no longer be part of the view and disappear from the view.

The situation may be avoided if you use WITH CHECK OPTION, which will restrict such updates. Thus, you may use the following option to create the view:

```

CREATE VIEW COSTLYITEM AS
SELECT *
FROM ITEM
WHERE ItemPrice > 15
WITH CHECK OPTION;

```

8.4.2 Sequences

Consider a situation, where in a column, you want to assign the next number in a sequence in every new record. Sequences are used for generating a new unique number of a sequence for a specific column. Sequences may be used for automatically generating the unique primary key values, as they can be generated efficiently. The following example explains the use of sequences:

Example 14: You can create a sequence, say ENRSEQ, for generating unique enrolment numbers of the students as follows (Please note different DBMS may have a different command for such sequences):

```
CREATE SEQUENCE ENRSEQ  
START WITH 230000001  
INCREMENT BY 1  
MAX VALUE 239999999;
```

The SQL commands, as given in example 14 will generate the sequence numbers ENRSEQ, but how will you use these sequences in tables? This is explained in example 15.

Example 15: This example demonstrates the use of sequence numbers while inserting records in a table. Consider a relation STUDENT (StudentID, StudentName, StudentProgCode). The StudentID is the primary key of the relation and should be unique and 9 characters long in the range 230000001 to 239999999. To automatically generate the sequence of student ids you may use the following command:

```
INSERT INTO STUDENT VALUES (ENRSEQ.NEXTVAL, "ABCD", "PGDCA");  
INSERT INTO STUDENT VALUES (ENRSEQ.NEXTVAL, "BCDE", "BCA");  
INSERT INTO STUDENT VALUES (ENRSEQ.NEXTVAL, "CDEF", "PGDCA");
```

You can display the content of the table after these insertions using the following commands:

```
SELECT * FROM STUDENT;
```

StudentID	StudentName	StudentProgCode
230000001	ABCD	PGDCA
230000002	BCDE	BCA
230000003	CDEF	PGDCA

You may please note that all the details of a sequence, like starting number, increment value and maximum values are stored and updated from time to time in the data dictionary. Further, certain situations like the rollback of an insert operation may result in gaps in the inserted values of a sequence.

Can you modify a sequence, once created? Yes, for this purpose, you may use the ALTER SEQUENCE statement of SQL. For example, you can change the increment value to 5 and the maximum value to 235000000 for the sequence ENRSEQ using the following command: the

```
ALTER SEQUENCE ENRSEQ  
START WITH 230000001
```

```
INCREMENT 5  
MAX VALUE 235000000;
```

You can also delete a sequence by giving the following command:

```
DROP SEQUENCE ENRSEQ;
```

8.4.3 Indexes and Synonyms

An index is part of the physical/internal schema of relational design. It is one of the important access structures of a database. Indexes help in enhancing the speed of the query processing. Every database has a basic set of indexes, like the primary key index, which are created automatically by the database management system. Additional indexes can be created by the database administrator, if needed.

You can create indexes using the following command:

Example 16: Consider the STUDENT relation, as given in Example 15. You need to create indexes for (i) To enhance the queries related to various programmes (ii) to list the students of a programme in the order of their names, for this case, you may like to create an index on programme and student name. You can use the following commands to create these indexes:

```
CREATE INDEX PROGCODE_IDX ON STUDENT (StudentProgCode);  
CREATE INDEX PROG_STUDENT_IDX ON STUDENT (StudentProgCode, StudentName);
```

It may be noted that the information of the indexes of the tables is stored in the data dictionary of a DBMS.

Further, you may note that though an index may help in improving query response time, they increase the time of data insertion, modification, and deletion, as these operations require updating the contents of related indexes. In general, you cannot modify the structure of an index, however, you can delete an index. To remove an index, you may give the command:

```
DROP INDEX PROGCODE_IDX;
```

Several DBMSs allow creation of alternative links or names to the database items or objects. These alternative names are generally short names and used for convenient references. For example, a typical DBMS may allow the creation of short names using the following command:

Example 17: You may create the following synonym for the STUDENT tables of example 15.

```
CREATE SYNONYM st  
FOR STUDENT;
```

You can remove a synonym by giving the following command:

```
DROP SYNONYM st;
```

Check Your Progress 3

- 1) Consider the following relations:

```
EMPLOYEE (EmployeeNo, EmployeeName, EmployeePhone, EmployeeBasicPay)
```

```
PROJECT(ProjectNo, ProjectName, ProjectDuration)
```

```
EMPLOYEEPROJECT (EmployeeNo, ProjectNo, EmpRoleInProject)
```

Create a detailed view of Projects consisting of Project number, project name, employee name of employees working on the project and his/her role in the project.

.....
.....
.....

- 2) Create a three-digit long odd number sequence.
-
.....

- 3) What are the advantages of creating indexes?
-
.....

8.5 SUMMARY

This unit introduces you to join queries and sub-queries. First, the unit explains the use of the JOIN statement in SQL. A JOIN statement uses two tables each of which has at least one attribute on the same domain. These attributes also called joining attributes, are used to join the tables using a conditional operator. In case, this conditional operator is equality, then the join operation is called the equi-join. In addition, if the joining attributes have the same name in both the tables, then only one of these attributes is included in the output. This type of join is called the Natural join. Outer joins are used when the records of the table, which do not participate in join operation are to be output too. Self-join is performed when one column values are to be related to the same or another column of the same table. The joining condition of self-join requires removing redundant records. The unit discusses these joins with the help of examples. Next, the unit discusses the nested queries. The unit details some of the simple operators that are used with sub-queries with the help of examples. This unit also discusses the correlated subqueries, which are generally expensive to execute. Finally, the unit provides details of some of the important concepts used in database management systems such as views, sequences and indexes.

You should practice these queries using a DBMS.

8.6 SOLUTIONS/ANSWERS

Check Your Progress 1

- 1) (a) The information about what parts have been supplied as a part of at least one order is available in the ORDERDETAILS table, it may be joined with the ITEM table to find the names of the parts.

```
SELECT DISTINCT ITEM.ItemID, ItemName  
FROM ITEM, ORDERDETAILS  
WHERE ITEM.ItemID = ORDERDETAILS.ItemID;
```

You can also use a subquery (explained in the next section) for this information:

```
SELECT ItemID, ItemName  
FROM ITEM  
WHERE ItemID IN (    SELECT DISTINCT ItemID  
                      FROM ORDERDETAIL  
                );
```

(b) The information about what orders are available is in the ORDERDETAILS table and information about the items is available in the ITEM table. Therefore, you need to join these two tables for the required information.

```
SELECT OrderID, ORDERDETAILS.ItemID, ItemName, ItemPrice  
FROM ORDERDETAILS, ITEM  
WHERE ORDERDETAILS.ItemID = ITEM. ItemID;
```

(c) The following query may result in the desired information:

```
SELECT OrderID, SUM(QuantityPurchased*ItemPrice)  
FROM ORDERDETAILS, ITEM  
WHERE ORDERDETAILS.ItemID = ITEM. ItemID  
GROUP BY OrderID;
```

2) It will require outer join:

```
SELECT ITEM.ItemID, ItemName, OrderID  
FROM ITEM LEFT OUTER JOIN ORDERDETAILS  
ON ITEM.ItemID = ORDERDETAILS.ItemID;
```

3) This will require a self-join.

```
SELECT od1.ItemID, od2.ItemID  
FROM ORDERDETAILS od1, ORDERDETAILS od2  
WHERE od1.OrderID = od2.OrderID AND od1.ItemID < od2.ItemID;
```

Check Your Progress 2

- 1) A subquery is a SELECT statement that is part of a main SELECT statement. You may put the SELECT statement in the FROM, WHERE or HAVING clauses of the main SELECT statement. Subqueries are useful when you want to use the set operators. They can also be used where a smaller relation can be returned and used more efficiently.
- 2) (a) Find the name of the projects, which have the least duration. (Can you write a more efficient query in this case?)

```
SELECT ProjectName  
FROM PROJECT  
WHERE ProjectNo IN  
(SELECT MIN(ProjectDuration)  
     FROM PROJECT );
```

- (b) Find the name and basic pay of employees who are working on more than one project.

```
SELECT EmployeeNo, EmployeeName, EmployeeBasicPay  
FROM EMPLOYEE
```

```
WHERE EmployeeNo IN
  (SELECT EmployeeNo)
    FROM EMPLOYEEPROJECT
      GROUP BY EmployeeNo
        HAVING Count(ProjectNO) > 2);
```

(c) Find the name of the employee who earns more than the average salary of all the employees.

```
SELECT EmployeeNo, EmployeeName, EmployeeBasicPay
  FROM EMPLOYEE
    WHERE EmployeeBasicPay >
      (SELECT avg (e.EmployeeBasicPay)
        FROM EMPLOYEE e);
```

- 3) (i) To find names of employees who are working on all projects.
(ii) To find names of the projects on which no employee is presently working.

Check Your Progress 3

1.
CREATE VIEW PROJECTDETAILS AS
SELECT p.ProjectNo, ProjectName, EmployeeName, EmpRoleInProject
 FROM EMPLOYEE e, EMPLOYEEPROJECT ep, PROJECT p
 WHERE e.EmployeeNo = ep.EmployeeNo AND ep.ProjectNo = p.ProjectNo;
2.
CREATE SEQUENCE ODDSEQ3
START WITH 101
INCREMENT BY 2
MAX VALUE 999;
3. Indexes are very useful for answering queries using non-key attributes. However, they enhance overheads in terms of maintaining indexes.

UNIT 9 ADVANCED SQL

Structure

- 9.0 Introduction
- 9.1 Objectives
- 9.2 Assertions and Views
 - 9.2.1 Assertions
 - 9.2.2 Views
- 9.3 Embedded SQL and Dynamic SQL
 - 9.3.1 Embedded SQL
 - 9.3.2 Cursors and Embedded SQL
 - 9.3.3 Dynamic SQL
 - 9.3.4 SQLJ
- 9.4 Stored Procedures and Triggers
 - 9.4.1 Stored Procedures
 - 9.4.2 Triggers
- 9.5 Advanced Features of SQL
- 9.6 Summary
- 9.7 Solutions/Answers

9.0 INTRODUCTION

The Structured Query Language (SQL) is a standard query language for database systems. It is considered one of the factors contributing to the success of commercial database management systems, primarily because of the availability of this standard language on most commercial database systems. We have already discussed about SQL in the previous two Units, where we discussed the SQL commands for data definition, data manipulation, data control, joins, sub-queries, etc.

In this unit, we provide details of some of the advanced features of Structured Query Language. We will discuss Assertions and Views, Triggers, Standard Procedures and Cursors. The concepts of embedded and dynamic SQL and SQLJ, which are used along with JAVA, have also been introduced. Some of the advanced features of SQL have been covered. We will provide examples in various sections rather than including a separate section of examples. The examples given here are closer to SQL3 standard yet may not be applicable to every commercial database management system. For more details, you may go through the documentation of DBMS that you may use for implementation of database.

9.1 OBJECTIVES

After going through this unit, you should be able to:

- define Assertions and explain how they can be used in SQL;
- explain the concept of views, SQL commands on views and updates on views;
- define and use Cursors;
- discuss Triggers and write stored procedures; and
- explain Dynamic SQL and SQLJ.

9.2 ASSERTIONS AND VIEWS

One of the major requirements in a database system is to define constraints on various tables. Some of these simple constraints can be specified as primary key, NOT

NULL, check value and range constraints. Such constraints can be specified while creating the table using the statement CREATE TABLE using the clauses like NOT NULL, PRIMARY KEY, UNIQUE, CHECK etc. Referential constraints can be specified with the help of foreign key constraints. However, there are some constraints, which may relate to more than one table. These are called assertions. In this section, we will discuss two important concepts that we use in database systems, viz., views and assertions. Assertions are general constraints, while views are virtual tables. Let us discuss them in more detail.

9.2.1 Assertions

Assertions are general constraints. For example, a hypothetical University does not admit students whose age is more than 25 years OR is more than the minimum age of the teacher at that University. Such general constraints can be implemented with the help of an assertion statement. The syntax of an assertion statement is given below:

Syntax:

```
CREATE ASSERTION <Name>
CHECK (<Condition>);
```

The assertion name helps in identifying the constraints specified by the assertion. These names can be used to modify or delete an assertion later. Assuming that the University has a STUDENT and one FACULTY table, the assertion on the age of students can be implemented as:

```
CREATE ASSERTION age-constraint
CHECK (NOT EXISTS (
    SELECT *
    FROM STUDENT s
    WHERE s.age > 25
    OR s.age > (
        SELECT MIN (f.age)
        FROM FACULTY f
    )));
```

An assertion, as above, is enforced on a database system by the database management system such that the constraints stated in the assertion are not violated. Assertions are checked whenever a related relation changes.

Example 1: Consider the following relations of a university:
FACULTY (code, name, age, basic_salary, medical_allowance, other_benefits)
MEDICAL CLAIM (code, claimdate, amount, comment)

Write an assertion for the University, which verifies that the total medical claims made by a faculty member in the financial year 2022-23 should not exceed his/her medical allowance.

```
CREATE ASSERTION Total_medical_claim
CHECK (NOT EXISTS (
    SELECT code, SUM (amount), MIN (medical_allowance)
    FROM (FACULTY NATURAL JOIN MEDICAL CLAIM)
    WHERE claimdate >= "01-04-2022" AND claimdate < "01-04-2023"
    GROUP BY code
    HAVING MIN(medical_allowance) < SUM(amount)
));
```

OR

```
CREATE ASSERTION Total_medical_claim
CHECK (NOT EXISTS (
    SELECT *
```

```

FROM FACULTY f
WHERE f.code IN
    (SELECT code, SUM(amount)
     FROM MEDICAL_CLAIM m
     WHERE claimdate >= "01-04-2022" AND claimdate < "01-04-2023"
       AND f.code=m.code
       AND f.medical-allowance<SUM(amount)
    )
));

```

Please analyse both the queries above. So, now you can create an assertion. But how can these assertions be used in database systems? The general constraints may be designed as assertions, which can be put into the stored procedures. Thus, any violation of an assertion may be detected.

9.2.2 Views

A view is a virtual table, which does not actually store data. Then what does it contain? A view is a query on the physical tables that store the data. The SQL command for creating views is explained with the help of an example.

Example 2: A student's database has the following tables:

```

STUDENT (name, enrolmentno, dateofbirth)
MARKS (enrolmentno, subjectcode, smarks)

```

For the database above a view can be created for a teacher, who is allowed to view only the performance of the student in his/her subject, let us say MCS207.

```
CREATE VIEW SUBJECT_PERFORMANCE AS
```

```

(SELECT s.enrolmentno, name, subjectcode, smarks
 FROM STUDENT s, MARKS m
 WHERE s.enrolmentno = m.enrolmentno AND
       subjectcode 'MCS207'
 ORDER BY s.enrolmentno;

```

A view can be dropped using a DROP statement as:

```
DROP VIEW SUBJECT_PERFORMANCE;
```

The physical table, which stores the data on which the statement of the view is written, is referred to as the base table. You can create views on two or more base tables by combining the data using JOIN. Thus, a view hides the logic of joining the tables from a user. You can also index the views to speed up the performance of query evaluation. Once a view has been created, it can be queried exactly like a base table.

For example:

```

SELECT *
FROM STUDENT_PERFORMANCE
WHERE smarks > 50;

```

How are the views implemented?

There are two strategies for implementing the views. These are:

- Query modification
- View materialisation.

In the query modification strategy, any query that is made on the view is modified to include the view-defining expression. For example, consider the view

STUDENT_PERFORMANCE. Consider a query on this view as: “The teacher of the course MCS207 wants to find the maximum and average marks in the course.” The query for this in SQL would be:

```
SELECT MAX(smarks), AVG(smarks)  
FROM SUBJECT_PERFORMANCE
```

Since SUBJECT_PERFORMANCE is itself a view the query will be modified automatically as:

```
SELECT MAX (smarks), AVG (smarks)  
FROM STUDENT s, MARKS m  
WHERE s.enrolmentno=m.enrolmentno AND subjectcode= “MCS207”;
```

However, this approach has a major disadvantage. Consider that you are repeatedly executing a complex query on a view in a large database system, the query modification will have to be performed for every execution of the query leading to inefficient utilisation of resources such as time and space.

The view materialisation strategy solves this problem by creating a temporary physical table for a view. However, this strategy is not useful in situations where database tables are frequently updated, as it will require frequent updating of the materialised views.

Can views be used for Data Manipulations?

Some views can be used during DML operations like INSERT, DELETE and UPDATE. When you perform DML operations on a view, such modifications are passed to the underlying base tables. However, not all views allow DML operations. Conditions for the view that may allow Data Manipulation are:

A view allows data updating if it fulfils the following conditions:

- 1) View Created from a single table:
 - The view allows the INSERT operation if the PRIMARY KEY column(s) and all the NOT NULL columns are included in the view.
 - The view should not be defined using any aggregate function or GROUP BY, HAVING or DISTINCT clauses. This is because any update on aggregated attributes or groups cannot be traced back to a single tuple of the base table. For example, consider a view AVGMARKS (coursecode, avgmark) created on a base table STUDENT (st_id, coursecode, marks). In the AVGMARKS view, changing the class average marks for coursecode “MCS207” to 50 from a calculated value of 40, cannot be accounted for a single tuple in the STUDENT base table, as the average marks are computed from the marks of all the Student tuples for that coursecode. Thus, this update will be rejected.
- 2) The views in SQL that are defined using joins are normally NOT updatable in general.
- 3) WITH CHECK OPTION clause of SQL checks the updatability of data from views, therefore, must be used with views through which you want to update.

Views and Security

Views are useful for the security of data. A view allows a user to use the data that is available through the view; thus, the hidden data is not made accessible. Access privileges can be given on views. Let us explain this with the help of an example. Consider the view that we have created for teacher: STUDENT_PERFORMANCE. You can grant privileges to a teacher whose name is ‘ABC’ as:

GRANT SELECT, INSERT, DELETE ON STUDENT_PERFORMANCE TO ABC
WITH GRANT OPTION;

Advanced
SQL

Please note that the teacher ABC has been given the rights to query, insert and delete the records on the given view. Please also note s/he is authorised to grant these access rights (WITH GRANT OPTION) to any other user. The access rights can be revoked using the REVOKE statement:

REVOKE ALL ON STUDENT_PERFORMANCE FROM ABC;

☞ Check Your Progress 1

- 1) Consider a constraint – the value of the age field of the student at a formal University should be between 17 years and 50 years. Would you like to write an assertion for this statement?

.....
.....
.....

- 2) Create a view for finding the average marks of the students in various subjects for the tables given in example 2.

.....
.....
.....

- 3) Can the view created in problem 2 be used to update subjectcode?

.....
.....
.....

9.3 EMBEDDED SQL AND DYNAMIC SQL

SQL commands can be entered through a standard SQL command level user interface. Such interfaces are interactive in nature and the result of a command is shown immediately. Such interfaces are very useful for those who have some knowledge of SQL and want to create a new type of query. However, in a database application where a naïve user wants to make standard queries, that too using GUI like interfaces, probably an application program needs to be developed. Such interfaces sometimes require the support of a programming language environment.

Please note that SQL normally does not support a full programming paradigm (although the latest SQL has full API support), which allows it a full programming interface. In fact, most of the application programs are seen through a programming interface, where SQL commands are put wherever database interactions are needed. Thus, SQL is embedded into programming languages like C, C++, JAVA, etc. Let us discuss the different forms of embedded SQL in more detail.

9.3.1 Embedded SQL

The embedded SQL statements can be put in the application program written in C++, Java or any other host language. These statements sometime may be called static. Why are they called static? The term ‘static’ is used to indicate that the embedded SQL commands, which are written in the host program, do not change automatically during the lifetime of the program. Thus, such queries are determined at the time of database application design. For example, a *query statement* embedded in C++ to determine the status of a booking of a ticket for a train will not change. However, this

query may be executed for many different tickets. Please note that it will only change the input parameters to the query, which are ticket number, train number, date of boarding, etc., and not the query itself.

But how is such embedding done? Let us explain this with the help of an example.

Example 3: Write a C program segment that prints the details of a student whose enrolment number is given as input.

Let us assume the relation:

STUDENT (enrolno:char(9), name:Char(25), phone:integer(12), prog_code:char(3))

The C Program may be as follows:

```
/* add proper include statements*/
/*declaration in C program */
EXEC SQL BEGIN DECLARE SECTION;
    Char enrolno[10], name[26], p_code[4];
    int phone;
    int SQLCODE;
    char SQLSTATE[6]
EXEC SQL END DECLARE SECTION;

/* The connection needs to be established with SQL*/
/* Write appropriate connection statements */

/* program segment for the required function */
printf ("enter the enrolment number of the student");
scanf ("% s", &enrolno);
EXEC SQL
    SELECT name, phone, prog_code INTO
        :name, :phone, :p_code
    FROM STUDENT
    WHERE enrolno = :enrolno;
If(SQLCODE == 0)
    printf ("%d, %s, %s, %s", enrolno, name, phone, p-code)
else
    printf ("Wrong Enrolment Number");
```

Please note the following points in the program above:

- The program is written in the host language ‘C’ and contains embedded SQL statements.
- Although in the program an SQL query (SELECT) has been added. You can embed any DML, DDL or view statements.
- The distinction between an SQL statement and a host language statement is made by using the keyword EXEC SQL; thus, this keyword helps in identifying the Embedded SQL statements.
- Please note that the statements including (EXEC SQL) are terminated by a semi-colon (;).
- As the data is to be exchanged between a host language and a database, there is a need for shared variables that are shared between the environments. Please note that enrolno[10], name[20], p_code[4] etc. are shared variables and declared in ‘C’. The colon (:) character in the SQL statement precedes the shared variables to distinguish them from the Table attributes.
- Please note that the shared host variables enrolno is declared to have char[10] whereas, an SQL attribute enrolno has only char[9]. Why? Because in ‘C’ conversion to a string includes a ‘\ 0’ as the end of the string.

- The type mapping between ‘C’ and SQL types is defined in the following table:

‘C’ TYPE	SQL TYPE
long	INTEGER
short	SMALLINT
float	REAL
double	DOUBLE
char [i+1]	CHAR (i)

- Please also note that these shared variables are used in the SQL statements of the program. They are prefixed with the colon (:) to distinguish them from database attribute and relation names. However, they are used without this prefix in any C language statement.
- Please also note that these shared variables have almost the same name (except p_code) as that of the attribute name of the database. The prefix colon (:) distinguishes whether we are referring to the shared host variable or an SQL attribute. Such similar names are a good programming convention as it helps in identifying the related attribute easily.
- Please note that the shared variables are declared between BEGIN DECLARE SECTION and END DECLARE SECTION and their type is defined in ‘C’ language.

Two more shared variables have been declared in ‘C’. These are:

- SQLCODE as int.
- SQLSTATE as char of size 6.
- These variables are used to communicate errors and exception conditions between the database and the host language program. The value 0 in SQLCODE means successful execution of SQL command. A value of the SQLCODE =100 means ‘no more data’. SQLCODE < 0 indicates an error. Similarly, SQLSTATE is a 5-char code the 6th character is for ‘\0’ in the host language ‘C’. Value “00000” in an SQLSTATE indicates no error. You can refer to DBMS/SQL standard in more detail for more information.
- In order to execute the required SQL command, connection with the database server need to be established by the program. For this, the following SQL statement is used:

```
CONNECT <name of the server> AS <name of the connection>
    AUTHORISATION <username, password>;
```

To disconnect, you can simply give the command:

```
DISCONNECT <name of the connection>;
```

You must check all these statements from the documentation of the commercial database management system, which you are using.

Explanation of SQL query in the given program: In the given SQL query, first, the given value of the enrolment number is transferred to the SQL attribute value, the query then is executed and the result, which may be a single tuple in this case, is transferred to shared host variables as indicated by the keyword INTO after the SELECT statement.

The SQL query runs as a standard SQL query except for the use of shared host variables. The rest of the C program has very simple logic and will print the data of the students whose enrolment number has been entered.

Please note that in this query, as the enrolment number is the primary key to the relation, only one tuple will be transferred to the shared host variables. But what will

happen if the execution of embedded SQL query results in more than one tuple? Such situations are handled with the help of a cursor. Let us discuss such queries in more detail.

9.3.2 Cursors and Embedded SQL

Let us first define the term ‘cursor’. The database server may allocate a portion of RAM for database interaction and internal processing. This portion may be used for query processing using SQL. This portion of RAM is also called the cursor. What should be the size of memory for the query processing? Ideally, the size of memory allotted for the query processing should be equal to the memory required to hold the query result. However, the available memory puts a constraint on the allotted size. Whenever a query results in several tuples, you can use a cursor to process the currently available tuples one by one. How? Let us explain the use of the cursor with the help of an example:

Since most of the commercial RDBMS architectures are client-server architectures, on the execution of an embedded SQL query, the resulting tuples are cached in the cursor. This operation is performed on the server. Sometimes, the cursor is opened by RDBMS itself – these are called **implicit cursors**. However, in embedded SQL you need to declare these cursors explicitly – these are called **explicit cursors**. Any cursor needs to have the following operations defined on them:

DECLARE – to declare the cursor.
OPEN AND CLOSE - to open and close the cursor.
FETCH – get the current records one by one till the end of tuples.

In addition, cursors have some attributes through which we can determine the state of cursor. These may be:

ISOPEN – It is true if the cursor is OPEN, otherwise false.
FOUND/NOT FOUND – It is true if a row is fetched successfully/not successfully.
ROWCOUNT – It determines the number of tuples in the cursor.

Let us explain the use of the cursor with the help of an example:

Example 4: Write a C program segment that inputs the final grade of the students of PGDCA programme.

Let us assume the relation:

```
STUDENT (enrolno:char(9), name:Char(25), phone:integer(12),
          prog_code:char(3)); grade: char(1));
```

The program segment is:

```
/* add proper include statements*/
/*declaration in C program */
EXEC SQL BEGIN DECLARE SECTION;
    Char enrolno[10], name[26], p_code[4], grade; /* grade is just one character*/
    int phone;
    int SQLCODE;
    char SQLSTATE[6]
EXEC SQL END DECLARE SECTION;
/* The connection needs to be established with SQL*/
/* Write appropriate connection statements */

/* program segment for the required function */
```

```

printf ("enter the programme code);
scanf ("%s, &p_code);
EXEC SQL DECLARE CURSOR FINALGRADE
    SELECT enrolno, name, phone, grade
    FROM STUDENT
    WHERE prog_code =:p_code
    FOR UPDATE OF grade;
EXEC SQL OPEN FINALGRADE;
EXEC SQL FETCH FROM FINALGRADE
    INTO :enrolno, :name, :phone, :grade;
WHILE (SQLCODE==0) {
    printf ("enter grade for enrolment number, "%s", enrolno);
    scanf ("%c", grade);
    EXEC SQL
        UPDATE STUDENT
        SET grade=:grade
        WHERE CURRENT OF FINALGRADE
    EXEC SQL FETCH FROM FINALGRADE
        INTO :enrolno, :name, :phone, :grade;
}
EXEC SQL CLOSE FINALGRADE;

```

- Please note that the declared section remains almost the same. The cursor is declared to contain the output of the SQL statement. Please notice that in this case, there will be many tuples of students database, which belong to a particular programme.
- The purpose of the cursor is also indicated during the declaration of the cursor.
- The cursor is then opened and the first tuple is fetched into the shared host variable followed by an SQL query to update the required record. Please note the use of CURRENT OF which states that these updates are for the current tuple referred to by the cursor.
- WHILE Loop is checking the SQLCODE to ascertain whether more tuples are pending in the cursor.
- Please note that the SQLCODE will be set by the last fetch statement executed just prior to the WHILE condition check.

How are these SQL statements compiled and error checked during embedded SQL?

The SQL pre-compiler performs the type checking of the various shared host variables to find any mismatches or errors on each SQL statement. It then stores the results in the SQLCODE or SQLSTATE variables.

Is there any limitation on these statically embedded SQL statements?

They offer only limited functionality, as the query must be known at the time of application development so that they can be pre-compiled in advance. However, many queries are not known at the time of development of an application; thus, you require dynamically embedded SQL that are discussed next.

9.3.3 Dynamic SQL

Dynamic SQL, unlike embedded SQL statements, is built at the run time and placed in a string in a host variable. The created SQL statements are then sent to the DBMS for processing. Dynamic SQL is generally slower than statically embedded SQL as they require complete processing including access plan generation during the run time.

However, they are more powerful than *embedded SQL* as they allow run-time application logic. The basic advantage of using dynamic embedded SQL is that you

need not compile and test a new program for a new query. Let us explain the use of dynamic SQL with the help of an example.

Example 5: Write a dynamic SQL interface that allows a student to get and modify permissible details about him/her. The student may ask for a subset of information also. Assume that the student database has the following relations.

STUDENT (enrolno, name, dob)
RESULT (enrolno, coursecode, marks)

In the table above, a student has access rights for accessing information on his/her enrolment number, but s/he cannot update the data. Assume that the username used by a student is his/her enrolment number.

A sample program segment is given below (please note that the syntax may change for different commercial DBMSs).

```
/* declarations in SQL */  
EXEC SQL BEGIN DECLARE SECTION;  
    char inputfields(50);  
    char tablename(10)  
    char sqlquerystring(200)  
EXEC SQL END DECLARE SECTION;  
  
printf ("Enter the fields you want to see \n");  
scanf ("%s", &inputfields);  
printf ("Enter the name of table STUDENT or RESULT");  
scanf ("%s", &tablename);  
sqlquerystring = "SELECT" + inputfields + " "  
    "FROM" + tablename  
    + "WHERE enrolno + :USER"  
/*Plus is used as a symbol for concatenation operator; in some DBMS it may be ||*/  
/* Assumption: the username is available in the host language variable USER*/  
  
EXEC SQL PREPARE sqlcommand FROM :sqlquerystring;  
EXEC SQL EXECUTE sqlcommand;
```

Please note the following points in the example above.

- The query can be entered completely as a string by the user.
- The query can be fabricated using a concatenation of strings. The program segment is language dependent and is not portable to other programming languages or environments.
- Query modification may be performed, keeping data security in mind.
- The query is prepared and executed using a suitable SQL EXEC command.

9.3.4 SQLJ

Till now we have discussed embedding SQL in C, can you embed SQL statements into JAVA Program? For inserting SQL statements in JAVA, you use SQLJ. In SQLJ, a preprocessor called SQLJ translator translates the SQLJ source file to a JAVA source file. The JAVA file is compiled and run on the database. The use of SQLJ improves the productivity and manageability of JAVA Code as:

- The code becomes somewhat compact.
- No run-time SQL syntax errors as SQL statements are checked at compile time.
- It allows sharing of JAVA variables with SQL statements. Such sharing is not possible otherwise.

SQLJ provides a standard form in which SQL statements can be embedded in the JAVA program. SQLJ statements always begin with a #sql keyword. These embedded SQL statements are of two categories – Declarations and Executable Statements.

Declarations have the following syntax:

```
#sql <modifier> context context_classname;
```

The executable statements have the following syntax:

```
#sql {SQL operation returning no output};
```

OR

```
#sql result = {SQL operation returning output};
```

Example 6: Write a JAVA function to print the student details of the student table, for the students who have been admitted in 2023 or later and whose names are like ‘As’.

Assuming that the first two digits of the 9-digit enrolment number represent the year of admission, the required input conditions may be:

- The enrolment number should be more than “23000000” and
- The name contains the substring “As”.

Please note that these input conditions will not be part of the Student Display function, rather will be used in the main () function that may be created separately by you. The following display function will accept the values as the parameters supplied by the main ().

```
public void DISPSTUDENT (String enrolno, String name, int phone)
{
    try {
        SelRowIter srows = null;
        # sql srows = {
            SELECT enrolno, name, phone
            FROM STUDENT
            WHERE enrolno > 230000000 AND name LIKE "As%"
        };
        while ( srows.next () ) {
            int enrolno = srows.enrolno ();
            String name = srows.name ();
            int phone = srows.phone ();
            System.out.println ("Enrollment_No = " + enrolno);
            System.out.println ("Name =" + name);
            System.out.println ("phone =" + phone);
        }
    } Catch (Exception e) {
        System.out.println (" error accessing database" + e.toString());
    }
}
```

☞ Check Your Progress 2

- 1) A University decided to enhance the marks of all the students in the subject MCS207 by 2. (Table to be used: RESULT (enrolno, coursecode, marks)). Write a segment of the embedded SQL program to do this processing.

- 2) What is dynamic SQL?

.....
.....
.....

- 3) Can SQL be embedded in JAVA?

.....
.....
.....

9.4 STORED PROCEDURES AND TRIGGERS

In this section, we will discuss some standard features that make commercial databases a strong implementation tool for information systems. These features are triggers and stored procedures. Let us discuss them in more detail in this section.

9.4.1 Stored Procedures

Stored procedures are collections of small programs that are stored in compiled form. A stored procedure is designed for the implementation of a specific function. For example, a company may have rules such as:

- A code (like an enrolment number) contains a check digit to check the validity of the code.
 - The date of change of any value may be recorded.

These rules may apply to every application that uses that code. Thus, instead of inserting them in the code of each application they may be put in a stored procedure and reused.

The use of procedure has the following advantages from the viewpoint of database application development.

- They help in removing SQL statements from the application program, thus making the program more readable and maintainable.
 - They run faster than SQL statements since they are already compiled in the database.

Stored procedures can be created using CREATE PROCEDURE statement in some commercial DBMS.

Syntax:

Syntax:
CREATE [or replace] PROCEDURE [user] <ProcedureName>
 [(argument datatype [, argument datatype]...)]

BEGIN

Host Language statements:

END.

Example 7: Consider a data entry application that allows entry of the enrolment number, first name and last name of a student. The application then combines the first and last name and enters the complete name in uppercase in the table STUDENT. The student table has the following structure:

The stored procedure for this application may be written (using an embedded programming language) as:

```

CREATE PROCEDURE studententry (
    enrolment  INOUT char (9);
    f_name      INOUT char (20);
    l_name      INOUT char (20);
BEGIN
    /* change all the characters to uppercase and trim the length */
    f_name = TRIM(UPPER(f_name));
    l_name = TRIM(UPPER(l_name));
    name = f_name || " " || l_name;
    INSERT INTO CUSTOMER
        VALUES (enrolment, name);
END;

```

INOUT used in the host language indicates that this parameter may be used both for the input and output of values in the database.

While creating a procedure, if you encounter errors, then you can use the ***show errors*** command. It shows all the errors encountered by the most recently created procedure object.

You can also write an SQL command to display errors. The syntax for finding an error in a commercial database is:

```

SELECT *
FROM USER_ERRORS
WHERE name ='procedure name' and type='PROCEDURE';

```

Procedures are compiled by the DBMS. However, if there is a change in the tables then the procedure needs to be recompiled. You can recompile the procedure explicitly using the following command:

```
ALTER PROCEDURE procedure_name COMPILE;
```

You can drop a procedure by using DROP PROCEDURE command.

9.4.2 Triggers

Triggers are somewhat like stored procedures except that they are activated automatically. When is a trigger activated? A trigger is activated on the occurrence of a particular event. What are these events that can cause the activation of triggers? These events may be database update operations like INSERT, UPDATE, DELETE etc. A trigger consists of these essential components:

- An event that causes its automatic activation.
- The condition that determines whether the event has called an exception.
- The action that is to be performed.

Triggers do not take parameters and are activated automatically, thus, are different to stored procedures on these accounts. Triggers are used to implement constraints among more than one table. Specifically, the triggers should be used to implement the constraints that are not implementable using referential integrity or constraints. An instance of such a situation may be when an update in one relation affects a few tuples in another relation. However, please note that you should not be over-enthusiastic for

writing triggers – if any constraint is implementable using declarative constraints such as PRIMARY KEY, UNIQUE, NOT NULL, CHECK, FOREIGN KEY, etc., then it should be implemented using those declarative constraints rather than triggers, primarily due to performance reasons.

You may write triggers that may execute once for each row in a transaction – called Row Level Triggers, or once for the entire transaction - called Statement Level Triggers. Remember that you must have proper access rights on the tables on which you are creating the trigger. The following is the syntax of triggers in one of the commercial DBMSs:

```
CREATE TRIGGER <trigger_name>
[BEFORE | AFTER|
<Event>
ON <tablename>
[WHEN <condition> | FOR EACH ROW]
<Declarations of variables, if needed, – may be used when creating trigger
using host language>
BEGIN
    <SQL statements OR host language SQL statements>
[EXCEPTION]
    <Exceptions if any>
END;
```

Let us explain the use of triggers with the help of an example:

Example 8: Consider the following relation of a student's database:

STUDENT(enrolno, name, phone)
RESULT (enrolno, coursecode, marks)
COURSE (course-code, c-name, details)

Assume that the marks are out of 100 in each course. The passing marks in a subject are 50. The University has a provision for 2% grace marks for the students who are failing marginally – that is, if a student has 48 marks, s/he is given 2 marks grace and if a student has 49 marks, then s/he is given 1 grace mark. Write the suitable trigger for this situation.

Please note the requirements of the trigger:

Event: UPDATE of marks
OR

INSERT of marks

Condition: When a student has 48 OR 49 marks

Action: Give 2 grace marks to the student having 48 marks and 1 grace mark to the student having 49 marks.

The trigger for this thus can be written as:

```
CREATE TRIGGER grace
AFTER INSERT OR UPDATE OF marks ON RESULT
WHEN (marks = 48 OR marks = 49)
UPDATE RESULT
    SET marks = 50;
```

We can drop a trigger using a DROP TRIGGER statement.

```
DROP TRIGGER trigger_name;
```

The triggers are implemented in many commercial DBMSs. Please refer to the documentation of the respective DBMS for more details.

9.5 ADVANCED FEATURES OF SQL

The latest SQL standards have enhanced SQL tremendously. Let us touch upon some of these enhanced features. More details on these would be available in the online sources of SQL.

SQL Interfaces: SQL also has good programming level interfaces. The SQL supports a library of functions for accessing a database. These functions are also called the Application Programming Interface (API) of SQL. The advantage of using an API is that it provides flexibility in accessing multiple databases in the same program irrespective of DBMS, while the disadvantage is that it requires more complex programming. The following are two common functions, called interfaces:

SQL/CLI (SQL – call level interface) is an advanced form of Open Database Connectivity (ODBC).

Java Database Connectivity (JDBC) – allows object-oriented JAVA to connect to multiple databases.

SQL support for object orientation: The latest SQL standard also supports object-oriented features. It allows the creation of abstract data types, nested relations, object identifiers etc.

Interaction with Newer Technologies: SQL provides support for XML (eXtended Markup Language) and Online Analytical Processing (OLAP) for data warehousing technologies.

☛ Check Your Progress 3

- 1) What is the stored procedure?

.....
.....
.....

- 2) Write a trigger that restricts updating of the STUDENT table outside the normal working hours/holiday.

.....
.....
.....

9.6 SUMMARY

This unit has introduced some important advanced features of SQL. The unit has also provided information on how to use these features.

An assertion can be defined as the general constraint on the state of a database. These constraints are expected to be always satisfied by the database. The assertions can be stored as stored procedures.

Views are the external schema windows of the data from a database. Views can be defined on a single table or multiple tables and help in automatic security of hidden

data. All the views cannot be used for updating data in the tables. You can query a view.

The embedded SQL helps in providing complete host language support to the functionality of SQL, thus making application programming somewhat easier. An embedded query can result in a single tuple, however, if it results in multiple tuples then you need to use cursors to perform the desired operation. Cursor is a sort of pointer to the area in the memory that contains the result of a query. The cursor allows sequential processing of these result tuples. The SQLJ is the embedded SQL for JAVA. Dynamic SQL is a way of creating queries through an application and compiling and executing them at the run time. Thus, it provides a dynamic interface and hence the name.

Stored procedures are precompiled procedures and can be invoked from application programs. Arguments can be passed to a stored procedure and can return values. A trigger is also like a stored procedure except that it is invoked automatically on the occurrence of a specified event.

You should refer to the further readings for more details on the topics relating to this unit.

9.7 SOLUTIONS/ANSWERS

Check Your Progress 1

- 1) The constraint is a simple Range constraint and can easily be implemented with the help of declarative constraint statement. (You need to use CHECK CONSTRAINT statement). Therefore, there is no need to write an assertion for this constraint.
- 2)

```
CREATE VIEW avgmarks AS (
    SELECT subjectcode, AVG(smarks)
    FROM MARKS
    GROUP BY subjectcode);
```
- 3) No, the view is using a GROUP BY clause. Thus, if you try to update the subjectcode, you cannot trace it back to a single tuple where such a change needs to take place.

Check Your Progress 2

- 1)

```
/*The table is RESULT (enrolno, coursecode, marks). */
EXEC SQL BEGIN DECLARE SECTION;
    char enrolno [10], coursecode[7];
    int marks;
    int SQLCODE;
    char SQLSTATE[6]
EXEC SQL END DECLARE SECTION;
/*The connection needs to be established with SQL*/
/* program segment for the required function*/
    printf ("enter the course code for which 2 marks are to be added");
    scanf ("%s", &coursecode);
EXEC SQL DECLARE CURSOR GRACE
    SELECT enrolno, coursecode, marks
    FROM RESULT
    WHERE coursecode=:coursecode
```

```

/* For update of marks */
EXEC SQL OPEN GRACE;
EXEC SQL FETCH FROM GRACE
    INTO :enrolno, :coursecode, :marks;
WHILE (SQL CODE==0)
{
    EXEC SQL
        UPDATE RESULT
        SET marks = marks+2
        WHERE CURRENT OF GRACE;
    EXEC SQL FETCH FROM GRACE
        INTO :enrolno, :coursecode, :marks;
}
EXEC SQL CLOSE GRACE;

```

An alternative implementation in a commercial database management system may be:

```

DECLARE CURSOR grace IS
    SELECT enrolno, coursecode, marks
    FROM RESULT
    WHERE coursecode ='MCS207';
    str_enrolno RESULT.enrolno%type;
    str_coursecode RESULT.coursecode%type;
    str_marks RESULT.marks%type;
BEGIN
    OPEN grace;
    IF GRACE %OPEN THEN
        LOOP
            FETCH grace INTO str_enrolno, str_coursecode, str_marks;
            Exit when grace%NOTFOUND;
            UPDATE student SET marks=str_marks +2;
            INSERT INTO resultmcs207 VALUES
                (str_enrolno, str_coursecode, str_marks);
        END LOOP;
        COMMIT;
        CLOSE grace;
    ELSE
        Dbms_output.put_line ('Unable to open cursor');
    END IF;
END;

```

- 2) Dynamic SQL allows run-time query-making through embedded languages. The basic step here would be - create a valid query string and then execute that query string. Since the queries are compiled and executed at the run time thus, it is slower than embedded SQL.
- 3) Yes. You may use SQLJ for this purpose.

Check Your Progress 3

- 1) Stored procedure is a compiled procedure in a host language that has been written for a specific purpose.
- 2) The trigger is some pseudo-DBMS may be written as:

```

CREATE TRIGGER resupdstudent
    BEFORE INSERT OR UPDATE OR DELETE ON STUDENT
BEGIN

```

```
IF (DAY (SYSDATE) IN ('SAT', 'SUN')) OR
(HOURS (SYSDATE) NOT BETWEEN '09:00' AND 18:30')
THEN
RAISE_EXCEPTION AND OUTPUT ('OPERATION NOT
ALLOWED AT THIS TIME/DAY');
END IF;
END;
```

Please note that we have used some hypothetical functions, the syntax of which will be different in different RDBMS.



UNIT 10 TRANSACTIONS AND CONCURRENCY MANAGEMENT

Structure	Page Nos.
10.0 Introduction	
10.1 Objectives	
10.2 The Transactions	
10.2.1 Properties of a Transaction	
10.2.2 States of a Transaction	
10.3 Concurrent Transactions	
10.3.1 Transaction Schedule	
10.3.2 Problems of Concurrent Transactions	
10.4 The Locking Protocol	
10.4.1 Serialisable Schedule	
10.4.2 Locks	
10.4.3 Two-Phase Locking (2PL)	
10.5 Deadlock Handling and its Prevention	
10.6 Optimistic Concurrency Control	
10.7 Timestamp-Based Protocols	
10.7.1 Timestamp Based Concurrency Control	
10.7.2 Multi-version Technique	
10.8 Weak Level of Consistency and SQL commands for Transactions	
10.9 Summary	
10.10 Solutions/ Answers	

10.0 INTRODUCTION

One of the main advantages of storing data in an integrated repository or a database is to allow sharing of data amongst multiple users. Several users access the database or perform transactions at the same time. What if a user's transactions try to access a data item that is being used /modified by another transaction? This unit attempts to provide details on how concurrent transactions are executed under the control of DBMS. However, in order to explain the concurrent transactions, first this unit describes the term transaction. Concurrent execution of user programs is essential for better performance of DBMS, as concurrent running of several user programs keeps utilising CPU time efficiently, since disk accesses are frequent and are relatively slow in case of DBMS. This unit not only explains the issues of concurrent transaction but also explains algorithms to control those problems.

10.1 OBJECTIVES

After going through this unit, you should be able to:

- define the term database transactions and their properties;
- describe the issues of concurrent transactions;
- explain the mechanism to prevent issues arising due to concurrently executing transactions;
- describe the principles of locking and serialisability; and
- describe concepts of deadlock and its prevention.

10.2 THE TRANSACTIONS

A transaction is a unit of data processing. Database systems that deal with a large number of concurrent transactions are also called Transaction Processing Systems. A

transaction is a unit of work in a database system. For example, a database system for a bank may allow the transactions such as – withdrawal of money from an account; deposit of money to an account; transfer of money from A's account to B's account. A transaction would involve the manipulation of one or more data values in a database. Thus, it may require reading and writing of database values. The following are examples of transactions.

Example 1: A money withdrawal transaction of a bank can be written in pseudo-code as:

; Assume that you are doing this transaction for account number X.

```
TRANSACTION WITHDRAWAL (withdrawal_amount)
Begin transaction
    IF X exists then
        READ X.balance
        IF X.balance > withdrawal_amount
            THEN
                SUBTRACT withdrawal_amount from X.balance
                WRITE X.balance
                Dispense withdrawal_amount
                COMMIT
            ELSE
                DISPLAY "TRANSACTION CANNOT BE PROCESSED"
            ELSE DISPLAY "ACCOUNT X DOES NOT EXIST"
        End transaction.
```

Another similar example may be the transfer of money from account number X to account number Y. This transaction may be written as:

Example 2: Transaction to transfers some amount from account X to account Y.

```
TRANSACTION (X, Y, transfer_amount)
Begin transaction
    IF X AND Y exist then
        READ X.balance
        IF X.balance > transfer_amount THEN
            X.balance = X.balance - transfer_amount
            READ y.balance
            Y.balance = Y.balance + transfer_amount
            COMMIT
        ELSE DISPLAY ("INSUFFICIENT BALANCE IN X")
            ROLLBACK
        ELSE DISPLAY ("ACCOUNT X OR Y DOES NOT EXIST")
    End transaction
```

Please note the following:

The two keywords here COMMIT and ROLLBACK are:

COMMIT makes sure that all the changes made by transactions are made permanent.

ROLLBACK terminates the transactions and rejects any change made by the transaction.

In general, databases are stored in secondary storage as data blocks, whereas they can be processed in the main memory. The portion of main memory allotted for database processing is called a buffer. When a transaction desires to update a value X, a transaction program performs the following operations:

Read the block containing value X to memory buffer

Process the value of X in the memory buffer

Write the value of X to the data block

A block of data on secondary storage may contain many data items, as its size may be in MBs. Further, a transaction processing system may modify several data items simultaneously, therefore, an interesting question for a DBMS is: when to write a data block back to secondary storage? This process is called Buffer management. You may refer to further readings for details on this topic.

Transactions have certain desirable properties. Let us look into the properties of a transaction.

10.2.1 Properties of a Transaction

A transaction has four basic properties. These are:

- Atomicity
- Consistency
- Isolation
- Durability

These are also called the ACID properties of transactions.

Atomicity: It defines a transaction to be a single unit of processing. In other words, either a transaction will be done *completely or not at all*. For example, in the transaction of Example 2, the transaction is reading and writing more than one data items. The atomicity property requires either operations on both the data item to be performed or not at all.

Consistency: This property ensures that complete transaction execution takes a database from one consistent state to another consistent state. If a transaction fails even then the database should come back to a consistent state, i.e., either to the database state that was before the start of the transaction or the database state after the end of the transaction.

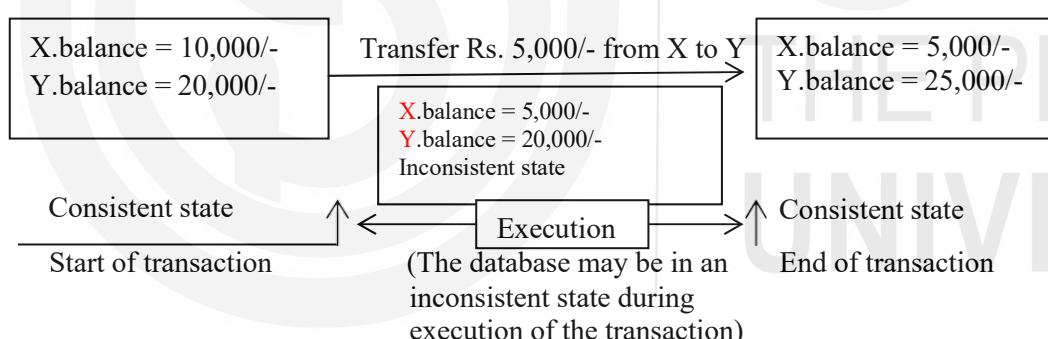


Figure 1: A Transaction execution

Isolation or Independence: The isolation property states that the updates of a transaction should not be visible till they are committed. Isolation guarantees that the progress of a transaction does not affect the outcome of another transaction. For example, if another transaction that is a withdrawal transaction which withdraws an amount of Rs. 5000 from X account is in progress, then failure or success of the transaction of example 2, should not affect the outcome of this transaction. Only the state of the data that has been read by the transaction should determine the outcome of this transaction.

Durability: This property necessitates that once a transaction has been committed, the changes made by it be never lost because of subsequent failure. Thus, a transaction is also a basic unit of recovery. The details of transaction-based recovery are discussed in the next unit.

10.2.2 States of a Transaction

A transaction can be in any one of the states during its execution. These states are displayed in *Figure 2*.

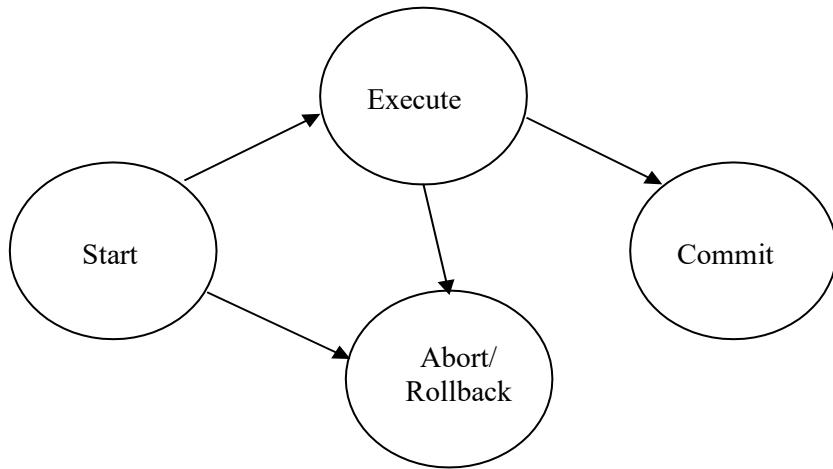


Figure 2: States of transaction execution

A transaction is started by a program. When a transaction is scheduled for execution by the Processor, it moves to the Execute state; however, in case of any system error at that point, it may be moved into the Abort state. During its execution, a transaction changes the data values, which may move the database to an inconsistent state. On successful completion of a transaction, it moves to the Commit state, where the durability feature of transaction ensures that the changes will not be lost. However, in case of an error in the execute state, the transaction goes to the Rollback state, where all the changes made by the transaction are undone. Thus, after commit or rollback, the database is back into consistent state. In case a transaction has been rolled back, it can be started as a new transaction. All these states of the transaction are shown in *Figure 2*.

10.3 THE CONCURRENT TRANSACTIONS

Almost all commercial DBMSs support multi-user environment, allowing multiple transactions to proceed simultaneously. The DBMS must ensure that two or more transactions do not get into each other's way, i.e., a transaction of one user does not affect the transactions of other users or even the other transactions issued by the same user. Please note that concurrency related problems may occur in databases only if **two transactions are contending for the same data item and at least one of the concurrent transactions wishes to update a data value in the database**. In case the concurrent transactions only read the same data item and no updates are performed on these values, then they do NOT cause any concurrency related problem. Now, let us first discuss why you need a mechanism to control concurrent transactions. This is explained next.

10.3.1 Transaction Schedule

Consider a banking application dealing with checking and saving accounts. A Banking Transaction T1 for Mr. Sharma moves Rs.100 from his checking account balance X to his savings account balance Y, using the transaction T1:

Transaction T1:

```
A:Read X  
Subtract
```

```

100
Write X
B:Read Y
Add 100
Write Y

```

Let us suppose an auditor wants to know the total assets of Mr. Sharma. S/he executes the following transaction:

Transaction T2:

```

Read X
Read Y
Display X+Y

```

Suppose both of these transactions are issued simultaneously, then the execution of these instructions can be mixed in many ways. This is also called the **Schedule**. Let us define this term in more detail.

Consider that a database system has n active transactions, namely T_1, T_2, \dots, T_n . Each of these transactions can be represented using a transaction program consisting of operations, as shown in Example 1 and Example 2. A schedule, say S , is defined as the sequential ordering of the operations of the ' n ' interleaved transactions, where the sequence or order of operations of an individual transaction is maintained.

Conflicting Operations in Schedule: Two operations of different transactions conflict if they access the same data item AND one of them is a write operation.

For example, the two transactions TA and TB, as given below, if executed in parallel, may produce a schedule:

TA
READ X
WRITE X

TB
READ X
WRITE X

(a) The two transactions

SCHEDULE	TA	TB
READ X	READ X	
READ X		READ X
WRITE X		WRITE X
WRITE X	WRITE X	

(b) One possible conflicting schedule for interleaved execution of TA and TB

Figure 3: A conflicting schedule

Let us show you three simple ways of interleaved instruction execution of transactions T1 and T2. Please note that in the following tables the first column defines the sequence of instructions that are getting executed, that is the schedule of operations.

Schedule	Transaction T1	Transaction T2	Example Values
Read X		Read X	X = 50000
Read Y		Read Y	Y = 100000
Display X+Y		Display X+Y	150000
Read X	Read X		X = 50000
Subtract 100	Subtract 100		= 49900
Write X	Write X		X = 49900

Read Y	Read Y		Y= 100000
Add 100	Add 100		100100
Write Y	Write Y		Y= 100100

- a) Complete execution of T2 is before T1 starts, then sum X+Y will show the correct assets.

Schedule	Transaction T1	Transaction T2	Example Values
Read X	Read X		X = 50000
Subtract 100	Subtract 100		49900
Write X	Write X		X = 49900
Read Y	Read Y		Y= 100000
Add 100	Add 100		100100
Write Y	Write Y		Y= 100100
Read X		Read X	X = 49900
Read Y		Read Y	Y= 100100
Display X+Y		Display X+Y	150000

- b) Complete execution of T1 is before T2 starts, then sum X+Y will still show the correct assets.

Schedule	Transaction T1	Transaction T2	Example Values
Read X	Read X		X = 50000
Subtract 100	Subtract 100		49900
Write X	Write X		X = 49900
Read X		Read X	X = 49900
Read Y		Read Y	Y= 100000
Display X+Y		Display X+Y	149900
Read Y	Read Y		Y= 100000
Add 100	Add 100		100100
Write Y	Write Y		Y= 100100

- c) Part execution of transaction T1, followed by the complete execution of T2, followed by the remaining part of T1.

Figure 4: Three different possibilities of interleaved execution of T1 and T2

In this execution, an incorrect value is displayed. This is because Rs.100, although removed from X, has not been put in Y and is thus missing from the computation of T2. Please note that for the given transaction there are many more ways of interleaved instruction execution.

Thus, there can be a possibility of anomalies when the transactions T1 and T2 are allowed to execute in parallel. Let us define the anomalies of concurrent execution of transactions more precisely.

10.3.2 Anomalies of Concurrent Transactions

Let us assume the following transactions (assuming there will not be errors in datawhile execution of transactions)

Transaction T3 and T4: T3 reads the balance of account X and subtracts a withdrawal amount of Rs. 5000, whereas T4 reads the balance of account X and adds an amount of Rs. 3000

T3
READ X
SUB 5000
WRITE X

T4
READ X
ADD 3000
WRITE X

The possible problems in the concurrent execution of these transactions are:

1. **Lost Update Anomaly:** Suppose the two transactions T3 and T4 run concurrently, and they happen to be interleaved in the following way (assume the initial value of X as 10000):

T3	T4	Value of X	
		T3	T4
READ X		10000	
	READ X		10000
SUB 5000		5000	
	ADD 3000		13000
WRITE X		5000	
	WRITE X		13000

After the execution of both transactions, the value X is 13000, while the semantically correct value should be 8000. The problem occurred as the update made by T3 has been overwritten by T4. The root cause of the problem was the fact that both the transactions had read the value of X as 10000. Thus, one of the two updates has been lost and we say that a **lost update** has occurred.

There is one more way in which the lost updates can arise. Consider the following part of some transactions T5 and T6; each of which increases the value of X by 1000.

T5	T6	Value of x originally 2000	
		T5	T6
Update X		3000	
	Update X		4000
ROLLBACK		2000	

Here, transactions T5 and T6 update the same item X (please note that an Update include Reading, Modifying and Write operations). Thereafter, T5 decides to undo its action and rolls back, causing the value of X to go back to the original value 2000. In this case, the update performed by T6 had got lost and a lost update is said to have occurred.

2. **Unrepeatable read Anomaly:** Suppose transaction T7 reads X twice during its execution. If it did not update X itself, it could be very disturbing to see a different value of X in its next read. But this could occur if, between the two read operations, another transaction modifies X.

T7	T8	Assumed value of X=2000	
		T7	T8
READ X		2000	
	Update X		3000
READ X		3000	

Thus, inconsistent values are read, and the results of the transaction may be in

error.

3. **Dirty Read Anomaly:** T10 reads a value which has been updated by T9. This update has not been committed and T9 aborts.

T9	T10	Value of x old value =200	
		T9	T10
Update X		500	
	READ X		500
ROLLBACK		200	?

Here T10 reads a value that has been updated by transaction T9 that has been aborted. Thus, T10 has read a value that would never exist in the database and hence the problem.

Please note that all three problems that we have discussed so far are primarily due to the violation of the Isolation property of the concurrently executing transactions that use the same data items.

In addition to the three anomalies, sometimes concurrent transactions may result in inconsistent analysis. For example, in the schedule of the two transactions T1 and T2, as shown in Figure 4(c), you may observe that one part of transaction T1 is executed prior to the execution of transaction T2 and the other part after the completion of Transaction T2. You may also note that transaction T2 produces an incorrect sum of the balance of accounts X and Y. The cause of the problem is that T2 transaction has accessed data items, which are still being modified by another transaction. The important thing to note is that the modifying transaction has been able to modify only some of the data values and some data values are yet to be modified. This resulted in a situation where the analysis transaction has read some modified values and some unmodified values, resulting in an inconsistent output.

As you can observe from the problems discussed in this section, concurrency-related problems occur when multiple transactions contend for a data item concurrently. But how do we ensure that the execution of two or more transactions does not result in concurrency-related problems?

Well, one of the commonest techniques used for this purpose is to restrict access to data items that are being read or written by one transaction and are also being written by another transaction. This technique is called locking. Let us discuss locking in more detail in the next section.

☞ Check Your Progress 1

- 1) What is a transaction? What are its properties? Can a transaction update more than one data value? Can a transaction write a value without reading it? Give an example of a transaction.

.....
.....
.....

- 2) What are the anomalies of concurrent transactions? Can these problems occur in transactions which do not read the same data values?

.....
.....

- 3) What is a Commit state? Can you rollback after the transaction commits?

10.4 THE LOCKING PROTOCOL

To control concurrency related problems, we use locking. A lock is basically a variable that is associated with a data item in the database. A lock can be placed by a transaction on a shared resource. A locked data item is available for the exclusive use of the transaction that has locked it. Other transactions are locked out of that data item. When a transaction that has locked a data item does not desire to use it anymore, it should unlock the data item so that other transactions can use it. If a transaction tries to lock a data item already locked by some other transaction, it cannot do so and waits for the data item to be unlocked. The component of DBMS that controls and manages the locking and unlocking of data items is called the Lock Manager. The locking mechanism helps us to convert a schedule into a serialisable schedule. We had defined what a schedule is, but what is a serialisable schedule? Let us discuss it in more detail in the next section.

10.4.1 Serialisable Schedule

If the operations of two transactions conflict with each other, how to determine that no concurrency-related problems have occurred in the transaction execution? For this purpose, let us define the term – Schedule, Serial Schedule, interleaved schedule and Serializable Schedule.

A schedule can be defined as a sequence of actions/operations of one or more transactions, as explained in section 10.3.1. Figure 5 shows two schedules, viz. Schedule A and Schedule B.

A serial schedule is one in which the actions/operations of one transaction are performed at a time. This is followed by the actions/operations of the next transaction and so on. For example, Schedule A and Schedule B of Figure 5 are serial schedules, as in no schedule operations of transactions T1 and Transaction T2 interleave with each other.

An interleaved schedule allows the interleaving of actions/operations of different transactions. For example, the schedule in Figure 6 is an interleaved schedule. The basic question here is: How will you find out that a given interleaved schedule has not resulted in concurrency-related problems?

Schedule A: T2 followed by T1		Schedule B: T1 followed by T2		
Schedule A	T1	T2	Schedule B	T1
READ X		READ X	READ X	READ X
READ Y		READ Y	SUBTRACT 100	SUBTRACT 100
DISPLAY X+Y		DISPLAY X+Y	WRITE X	WRITE X
READ X	READ X		READ Y	READ Y
SUBTRACT 100	SUBTRACT 100		ADD 100	ADD 100
WRITE X	WRITE X		WRITE Y	WRITE Y
READ Y	READ Y		READ X	
ADD 100	ADD 100		READ Y	READ Y
WRITE Y	WRITE Y		DISPLAY X+Y	DISPLAY X+Y

Figure 5: Serial Schedule of two transactions

Schedule C: An Interleaved Schedule		
Schedule	T1	T2
READ X	READ X	
SUBTRACT 100	SUBTRACT 100	
READ X		READ X
WRITE X	WRITE X	
READ Y		READ Y
READ Y	READ Y	
ADD 100	ADD 100	
DISPLAY X+Y		DISPLAY X+Y
WRITE Y	WRITE Y	

Figure 6 (a): An Interleaved Schedule

Schedule D: A non-conflict equivalent schedule of Schedule C		
Schedule	T1	T2
READ X	READ X	
SUBTRACT 100	SUBTRACT 100	
WRITE X	WRITE X	
READ X		READ X
READ Y		READ Y
READ Y	READ Y	
ADD 100	ADD 100	
DISPLAY X+Y		DISPLAY X+Y
WRITE Y	WRITE Y	

Figure 6 (b): An Interleaved Schedule

You may observe that the Schedule C in Figure 6 is an interleaved schedule. It has conflicting operations, like reading and writing X and Y. There can be many interleaved schedules of transactions T1 and T2. How do you identify which two schedules are equivalent? Well, for that, let us define the term Conflict Equivalence.

Conflict Equivalence: Two schedules are defined as conflict equivalent if any two conflicting operations in the two schedules are in the same order. For example, Schedule D, as given below, is not a conflict equivalent schedule of Schedule C, as the sequence of WRITE X by T1 and READ X by T2 are not in the same order in the two interleaved schedules.

Conflict Serialisability or Serialisability

A schedule is called conflict serialisable if it is conflict equivalent to a serial schedule. In other words, for a given interleaved schedule, if the sequence of read and write is in the same order as that of any one of the serial schedules, then the interleaved schedule is called a conflict serialisable or serialisable schedule. In case an interleaved schedule is not serialisable, then it may result in problems due to concurrent execution of transactions.

Any schedule that produces the same results as a serial schedule is called a serialisable schedule. The basis of serialisability is taken from the notion of a serial schedule. Considering there are two concurrent transactions, T1 and T2, how many different serial schedules are possible for these two transactions? The possible serial schedules for two transactions, T1 and T2, are:

T1 followed by T2 OR T2 followed by T1.

Likewise, the number of possible serial schedules with three concurrent transactions would be defined by the number of possible permutations, which are:

T1-T2-T3	T1-T3-T2	T2-T1-T3
T2-T3-T1	T3-T1-T2	T3-T2-T1

But how can a schedule be determined to be serialisable or not? Is there any algorithmic way of determining whether a schedule is serialisable or not?

Using the notion of precedence graph, an algorithm can be devised to determine whether an interleaved schedule is serialisable or not. In this graph, the transactions of the schedule are represented as the nodes. This graph also has directed edges. An edge from the node representing transaction T_i to node T_j means that there exists a **conflicting operation** between T_i and T_j and T_i precedes T_j in some conflicting operations. The basic principle for determining a serialisable schedule is that the precedence graph, which determines the sequences of occurrence of transactions, does not contain a cycle. Given a precedence graph with no cycles, then it must be equivalent to a serial schedule. The steps of constructing a precedence graph are:

1. Create a node for every transaction in the schedule.
2. Find the precedence relationships in conflicting operations. Conflicting operations are (read-write) or (write-read) or (write-write) on the same data item in two different transactions. But how to find them?
 - 2.1 For a transaction T_i , which *reads* an item A, find a transaction T_j that *writes* A later in the schedule. If such a transaction is found, draw an edge from T_i to T_j .
 - 2.2 For a transaction T_i , which has *written* an item A, find a transaction T_j later in the schedule that *reads* A. If such a transaction is found, draw an edge from T_i to T_j .
 - 2.3 For a transaction T_i which has *written* an item A, find a transaction T_j that *writes* A later than T_i . If such a transaction is found, draw an edge from T_i to T_j .
3. If there is any cycle in the graph, the schedule is not serialisable, otherwise, find the equivalent serial schedule of the transaction by traversing the transaction nodes starting with the node that has no input edge.

Let us explain the algorithm with the help of the following example.

Example:

Let us use this algorithm to check whether the schedule given in Figure 6(a) is Serialisable. Figure 7 shows the required graph. Please note as per step 1, we draw the two nodes for T_1 and T_2 . In the schedule given in Figure 6, please note that the transaction T_2 reads data item X, which is subsequently written by T_1 , thus there is an edge from T_2 to T_1 (clause 2.1). Also, T_2 reads data item Y, which is subsequently written by T_1 , thus there is an edge from T_2 to T_1 (clause 2.1). However, that edge already exists, so we do not need to redo it. Please note that there are no cycles in the graph, thus, the schedule given in Figure 6 is **serialisable**. The equivalent serial schedule (as per step 3) would be T_2 followed by T_1 , which is serial Schedule-A of Figure 5.

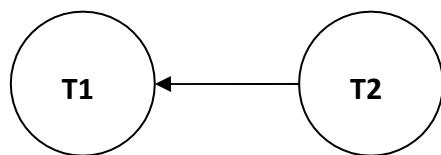


Figure 7: Test of Serialisability for the Schedule of Figure 6(a)

Please note that the schedule given in Figure 6(b) is not serialisable, because in that schedule, the two edges that exist between nodes T_1 and T_2 are:

- T_1 writes X which is later read by T_2 (clause 2.2), so there exists an edge from T_1 to T_2 .
- T_2 reads Y which is later written by T_1 (clause 2.1), so there exists an edge

from T2 to T1.

Thus, the graph for the schedule will be:

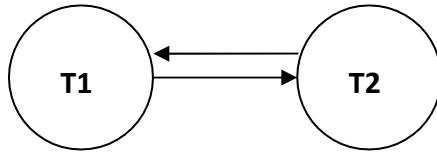


Figure 8: Test of Serialisability for the Schedule (c) of section 2.3

Please note that the graph above has a cycle T1-T2-T1, therefore it is **not serialisable**.

10.4.2 Locks

Serialisability is just a test of whether a given interleaved schedule is **serialisable** or has a concurrency related problem. However, it does not ensure that the interleaved concurrent transactions do not have any concurrency related problems. This can be done by using locks. So let us discuss what the different types of locks are, and then how locking ensures serialisability of executing transactions.

Types of Locks

There are two basic types of locks:

- Binary lock: This locking mechanism has two states for a data item: locked or unlocked.
- Multiple-mode locks: In this locking type each data item can be in three states read locked or shared locked, write locked or exclusive locked or unlocked.

Let us first take an example for binary locking and explain how it solves the concurrency related problems. Let us reconsider the transactions T1 and T2 and schedule given in Figure 4 (c) for this purpose; however, we will add required binary locks to them.

Schedule	T1	T2
LOCK X	LOCK X	
READ X	READ X	
SUBTRACT 100	SUBTRACT 100	
WRITE X	WRITE X	
UNLOCK X	UNLOCK X	
LOCK X		LOCK X
LOCK Y		LOCK Y
READ X		READ X
READ Y		READ Y
DISPLAY X+Y		DISPLAY X+Y
UNLOCK X		UNLOCK X
UNLOCK Y		UNLOCK Y
LOCK Y	LOCK Y	
READ Y	READ Y	
ADD 100	ADD 100	
WRITE Y	WRITE Y	
UNLOCK Y	UNLOCK Y	

Figure 9: An incorrect locking implementation

Does the locking as done above solve the problem of concurrent transactions? No, the same problems remain. Try working with the old values given in figure 4(c). Thus, locking should be done with some logic to make sure that locking results in no

concurrency related problem. One such solution is given below:

Schedule	T1	T2
LOCK X	LOCK X	
LOCK Y	LOCK Y	
READ X	READ X	
SUBTRACT 100	SUBTRACT 100	
WRITE X	WRITE X	
LOCK X (ISSUED BYT2)	LOCK X: denied as T1 holds the lock. The transaction T2 Waits and T1 continues.	
READ Y	READ Y	
ADD 100	ADD 100	
WRITE Y	WRITE Y	
UNLOCK X	UNLOCK X	
	LOCK X request of T2 on X will be granted and transaction T2 resumes.	
UNLOCK Y	UNLOCK Y	
LOCK Y		LOCK Y
READ X		READ X
READ Y		READ Y
DISPLAY X+Y		DISPLAY X+Y
UNLOCK X		UNLOCK X
UNLOCK Y		UNLOCK Y

Figure 10: A correct but restrictive locking implementation.

As shown in Figure 10, when you obtain all the locks at the beginning of the transaction and release them at the end, it ensures that transactions are executed with no concurrency-related problems. However, such a scheme limits the concurrency. We will discuss a two-phase locking method in the next subsection that provides sufficient concurrency. However, let us first discuss multiple-mode locks.

Multiple-mode locks: It offers two locks: shared locks and exclusive locks. But why do we need these two locks? There are many transactions in the database system that never update the data values. These transactions can coexist with other transactions that update the database. In such a situation multiple reads are allowed on a data item, so multiple transactions can lock a data item in the shared or read lock. On the other hand, if a transaction is an updating transaction, that is, it updates the data items, it must ensure that no other transaction can access (read or write) those data items that it wants to update. In this case, the transaction places an exclusive lock on the data items. Thus, a higher level of concurrency can be achieved compared to the binary locking scheme. The properties of shared and exclusive locks are summarised below:

a) Shared lock

- It is requested by a transaction that wants to just read the value of the data item.
- A shared lock on a data item does not allow an exclusive lock to be placed but permits any number of shared locks to be placed on that item.

b) Exclusive lock

- It is requested by a transaction on a data item that it needs to update.
- No other transaction can place either a shared lock or an exclusive lock on a data item that has been locked in an exclusive mode.

We explain these locks with the help of an example. However, you may refer to further readings for more information on multiple-mode locking. We will once again consider

the transactions T1 and T2, but in addition, a transaction T11 that finds the total of accounts Y and Z.

Schedule	T1	T2	T11
S_LOCK X		S_LOCK X	
S_LOCK Y		S_LOCK Y	
READ X		READ X	
S_LOCK Y			S_LOCK Y
S_LOCK Z			S_LOCK Z
			READ Y
			READ Z
X_LOCK X	X_LOCK X. The exclusive lock request on X is denied as T2 holds the Shared lock. The transaction T1 Waits.		
READ Y		READ Y	
DISPLAY X+Y		DISPLAY X+Y	
UNLOCK X		UNLOCK X	
X_LOCK Y	X_LOCK Y. The previous exclusive lock request on X is granted as X is unlocked. But the new exclusive lock request on Y is not granted as Y is locked by T2 and T11 in Shared mode. Thus, T1 waits till both T2 and T11 will release the Shared lock on Y.		
DISPLAY Y+Z			DISPLAY Y+Z
UNLOCK Y		UNLOCK Y	
UNLOCK Y			UNLOCK Y
UNLOCK Z			UNLOCK Z
READ X	READ X		
SUBTRACT 100	SUBTRACT 100		
WRITE X	WRITE X		
READ Y	READ Y		
ADD 100	ADD 100		
WRITE Y	WRITE Y		
UNLOCK X	UNLOCK X		
UNLOCK Y	UNLOCK Y		

Figure 11: Example of Locking in multiple modes

As shown in Figure 11, locking can result in a serialisable schedule. Next, we discuss the concept of granularity of locking.

Granularity of Locking

In general, locks may be allowed on the following database items:

- 1) The complete database itself
- 2) A file of the database
- 3) A page or disk block of data
- 4) A record in a table
- 5) A data item of an attribute

Granularity of locking depends on the size of database item being locked. A coarse granularity locking means locking a larger data item, e.g. the complete database or file or page etc. The fine granularity locking is locking the database items of the smaller size, e.g. the record locking or the data item locking.

A lock can be implemented as a binary variable, which requires space. In addition, a lock requires locking and unlocking operations. Thus, coarse locking has low locking overheads, but it restricts the concurrency among transactions. For example, if a single transaction locks the complete database in an exclusive mode, all other transactions will wait for the completion of this transaction. On the other hand, the fine granularity of locking results in high locking overheads.

To make an efficient concurrent transaction processing system, multiple granularity locking is supported by a database system. These system support Intention mode locking. You may refer to the further readings for more details on such locking scheme.

In the next section, we discuss the locking protocol that ensures correct execution of concurrent transactions.

10.4.3 Two-Phase Locking (2PL)

In this section we try to answer the question: Can you release locks a bit early and still have no concurrency related problem? Yes, we can do it if we use two-phase locking protocol. The two-phase locking protocol consists of two phases:

Phase 1: The lock acquisition phase: If a transaction T wants to read an object, it needs to obtain the S (shared) lock. If transaction T wants to modify an object, it needs to obtain X (exclusive) lock. No conflicting locks are granted to a transaction. **New locks on items can be acquired but no lock can be released, till all the locks required by the transaction are obtained.**

Phase 2: Lock Release Phase: The existing locks can be released in any order, but no new lock can be acquired **after a lock has been released**. The locks are held only till they are required.

Normally, any legal schedule of transactions that follows two-phase locking protocol is guaranteed to be serialisable. The two-phase locking protocol has been proven for its correctness. However, the proof of this protocol is beyond the scope of this Unit. You can refer to further readings for more details on this protocol.

There are two types of 2PL:

- (1) Conservative 2PL
- (2) Strict 2PL

The conservative 2PL allows the release of the lock at any time after all the locks have been acquired. For example, you can release the locks in the schedule of *Figure 10*, after you have read the values of Y and Z in transaction 11, even before the display of the sum. This will enhance the concurrency level. The conservative 2PL is shown graphically in *Figure 12*.

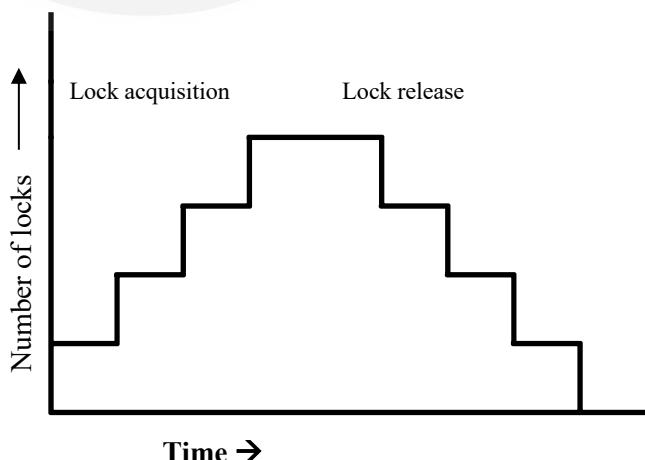


Figure 12: Conservative Two-Phase Locking

However, conservative 2PL suffers from the problem that it can result in loss of atomic or isolation property of transaction as theoretically speaking, once a lock is released on

a data item, it can be modified by another transaction before the first transaction commits or aborts.

To avoid such a situation, you use strict 2PL. The strict 2PL is graphically depicted in *Figure 13*. However, the basic disadvantage of strict 2PL is that it restricts concurrency as it locks the item beyond the time it is needed by a transaction.

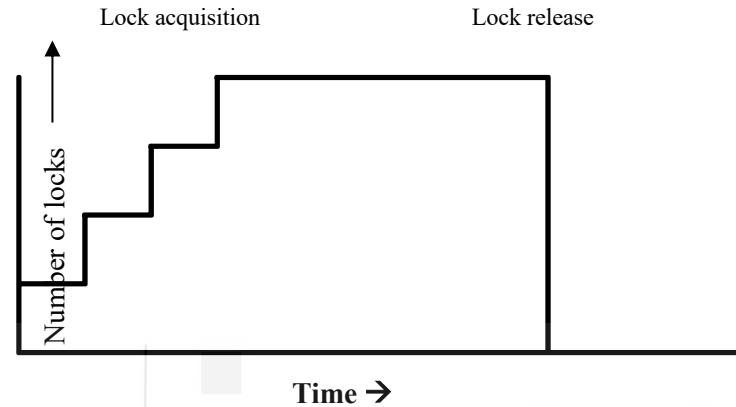


Figure 13: Strict Two-Phase Locking

Does the 2PL solve all the problems of concurrent transactions? No, the strict 2PL solves the problem of concurrency and atomicity; however, it introduces another problem: “Deadlock”. Let us discuss this problem in the next section.

☞ Check Your Progress 2

- 1) Let the transactions T1, T2, T3 perform the following operations:

T1: Add one to A

T2: Double A

T3: Display A on the screen and set A to one.

Suppose transactions T1, T2, and T3 are allowed to execute concurrently. If A has an initial value of zero, how many possible correct results are there? Enumerate them.

.....
.....

- 2) Consider the following two transactions on two bank accounts having a balance A and B.

Transaction T1: Transfer Rs. 100 from A to B

Transaction T2: Find the multiple of A and B.

Create a non-serialisable schedule.

.....
.....

- 3) Add lock and unlock instructions (exclusive or shared) to transactions T1 and T2 so that they observe the serialisable schedule. Make a valid schedule.

10.5 DEADLOCK HANDLING AND ITS PREVENTION

As seen earlier, though the 2PL protocol handles the problem of serialisability, but it causes some problems also. For example, consider the following two transactions and a schedule involving these transactions:

TA	TB
X_LOCK A	X_LOCK A
X_LOCK B	X_LOCK B
:	:
:	:
UNLOCK A	UNLOCK A
UNLOCK B	UNLOCK B

Schedule

```
T1: X_LOCK A  
T2: X_LOCK B  
T1: X_LOCK B  
T2: X_LOCK A
```

As is clearly seen, the schedule causes a problem. After T1 has locked A, T2 locks B and then T1 tries to lock B, but is unable to do so and waits for T2 to unlock B. Similarly, T2 tries to lock A but finds that it is held by T1 which has not yet unlocked it and thus waits for T1 to unlock A. At this stage, neither T1 nor T2 can proceed since both transactions are waiting for the other to unlock the locked resource (refer to figure 14).

Clearly, the schedule comes to a halt in its execution. The important thing to be seen here is that both TA and TB follow the 2PL, which guarantees serialisability. Whenever the situation, i.e. all the transactions are **waiting for a condition that will never occur**, arises, we say that a deadlock has occurred.

The deadlock can be described in terms of a directed graph called a “*wait for*” graph, which is maintained by the lock manager of the DBMS. This graph G is defined by the pair (V, E). It consists of a set of vertices/nodes V and a set of edges/arcs E. Each transaction is represented by node and an arc from $T_i \rightarrow T_j$, if T_j holds a lock on data items that T_i is waiting for. When transaction T_i requests a data item currently being held by transaction T_j then the edge $T_i \rightarrow T_j$ is inserted in the “*wait for*” graph. This edge is removed only when transaction T_j is no longer holding the data item needed by transaction T_i .

A deadlock in the system of transactions occurs, if and only if the wait-for graph contains a cycle. Each transaction involved in the cycle is said to be deadlocked.

To detect deadlocks, a periodic check for cycles in “*wait-for*” graph can be done. For example, the “*wait-for*” for the schedule of transactions TA and TB, as given earlier in this section, is shown in Figure 14.

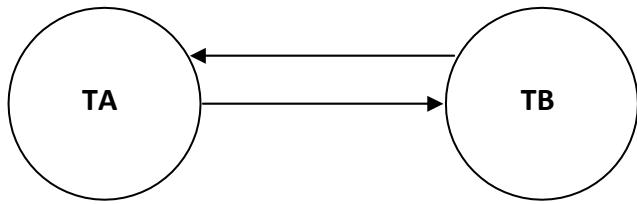


Figure 14: Wait For Graph of TA and TB

In Figure 14, TA and TB are the two transactions. The two edges are present between nodes TA and TB since each is waiting for the other to unlock a resource held by the other, forming a cycle and causing a deadlock problem. The above case shows a direct cycle. However, in actual situations, more than two nodes may be there in a cycle.

A deadlock is a situation that can be created because of locks. It causes transactions to wait forever hence the name deadlock. A deadlock occurs because of the following conditions:

- a) Mutual exclusion: A resource can be locked in exclusive mode by only one transaction at a time.
- b) Non-preemptive locking: A data item can only be unlocked by the transaction that locked it. No other transaction can unlock it.
- c) Partial allocation: A transaction can acquire locks on a database in a piecemeal way.
- d) Circular waiting: Transactions lock part of the data resources needed and then wait indefinitely to lock the resource currently locked by other transactions.

In order to prevent a deadlock, one has to ensure that at least one of these conditions does not occur.

A deadlock can be prevented, avoided, or controlled. Let us discuss a simple method for deadlock prevention. You can refer to other methods from the further readings.

Deadlock Prevention

One of the simplest approaches for avoiding a deadlock would be to acquire all the locks at the start of the transaction. However, this approach restricts concurrency greatly, also you may lock some of the items that are not updated by that transaction. A deadlock prevention algorithm prevents a deadlock. It uses the approach: *do not allow circular wait*. This approach rolls back some of the transactions instead of letting them wait.

There exist two such schemes. These are:

“Wait-die” scheme: The scheme is based on non-preventive technique. It is based on a simple rule:

If T_i requests a database resource that is held by T_j
then if T_i has a smaller timestamp than that of T_j
it is allowed to wait;
else T_i aborts.

A timestamp may loosely be defined as the system-generated sequence number that is unique for each transaction. Thus, a smaller timestamp means an older transaction. For example, assume that three transactions T_1 , T_2 and T_3 were generated in that sequence; then if T_1 requests for a data item which is currently held by transaction T_2 ,

it is allowed to wait as it has a smaller time stamping than that of T1. However, if T3 requests for a data item that is currently held by transaction T2, then T3 is rolled back (die).

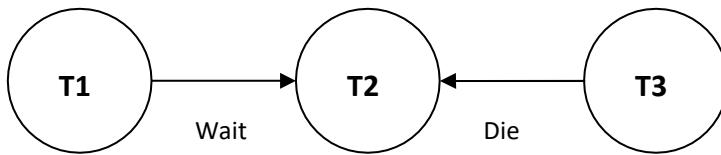


Figure 15: Wait-die Scheme of Deadlock Prevention.

“Wound-wait” scheme: It is based on a preemptive technique and is explained with the help of the following sequence of actions:

Resource Requesting Transaction	Resource is with the Transaction	Check Timestamp (TSP)	Action Needed
Ti	Tj	TSP (Ti) > TSP (Tj)	Ti is younger, therefore, can WAIT
		TSP (Ti) < TSP (Tj)	Ti is older, therefore, WOUND Ti

For example, consider three transactions T1, T2 and T3 with $\text{TSP}(T1) < \text{TSP}(T2) < \text{TSP}(T3)$, i.e. T1 is the oldest and T3 is the youngest. For example, consider the following sequence of requests for a data item X:

- | | |
|-------------------|---|
| T2 requests for X | Action: Locks the Resource |
| T1 requests for X | Action: Wound (Rollback) T1, as it is older. |
| T3 requests for X | Action: T3 Waits for T2 to release X, as it is younger. |

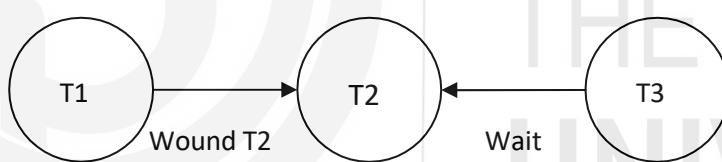


Figure 15: Wound-wait Scheme of Deadlock Prevention.

It is important to check that no transaction should get rolled back repeatedly such that it is never allowed to make progress. This is referred to starvation of a transaction. Also, both “wait-die” and “wound-wait” scheme should avoid starvation. The number of aborts and rollbacks will be higher in wait-die scheme than in the wound-wait scheme. But one major problem with both schemes is that these schemes may result in unnecessary rollbacks. You can refer to further readings for more details on deadlock-related schemes.

10.6 OPTIMISTIC CONCURRENCY CONTROL

Is locking the only way to prevent concurrency-related problems? There exist some other methods too. One such method is called an Optimistic Concurrency control. Let us discuss it in more detail in this section.

In general, the concurrency-related problem does not occur frequently, therefore, an

optimistic concurrency control scheme allows database transactions to freely perform the update operations on data items. Further, in the validation phase a transaction cross-checks if any other transaction has modified the data items, which it had read or written. Therefore, the optimistic concurrency control algorithm has the following phases:

- a) **READ Phase:** In this phase, a transaction makes a local copy of the data items needed by it in its own memory space. The transaction then makes changes in the local copies of data items.
- b) **VALIDATE Phase:** In this phase, the transaction validates the values of data items read by the transaction during the READ phase. This validation ensures that no other transaction has changed the values of the data items, while this transaction was modifying the local copy of the data items. In case, the validation is unsuccessful, then this transaction is aborted, and the local updates of data items are discarded. However, in case of successful validation, the Write phase is performed.
- c) **WRITE Phase:** This phase is performed if the Validate Phase is successful. In this phase, the transaction commits and all the data item updates made by the transaction in the local copies, are applied to the database.

To explain the optimistic concurrency control, the following terms are used:

- **Write set (WS(T)):** Given a transaction T, WS(T) is the set of data items that would be written by it.
- **Read set (RS(T)):** Given a transaction T, RS(T) is the set of data items that would be read by it.

More details on this scheme are available in further readings. But let us show this scheme here with the help of the following examples: Consider the set for transactions T1 and T2.

T1		T2	
Phase	Operation	Phase	Operation
-	-	Read	Reads the RS(T2), say variables X and Y and performs updating of local values
Read	Reads the RS(T1), say variable A and B and performs updating of local values	-	-
Validate	Validate the values of RS(T1)	-	-
-	-	Validate	Validate the values of RS(T2)
Write	Transaction Commits and writes the updated values WS(T1) in the database.	-	-
-	-	Write	Transaction Commits and writes the updated values WS(T2) in the database.

In this example, both T1 and T2 commit. Please note that read sets RS(T1) and RS(T2) are disjoint, also the Write sets are also disjoint, thus, no concurrency-related problem occurs.

Now let us consider another example, as given below:

T1	T2	T3
Read A	--	--
--	Read A	--
--	--	Read D
--	--	Update D
--	--	Update A
--	--	Validate (D, A) finds OK
--	--	Write (D, A), COMMIT
--	Validate (A): Unsuccessful Value changed by T3	--
Validate (A): Unsuccessful Value changed by T3	--	--
ABORT T1	--	--
--	ABORT T2	--

In this case, both T1 and T2 get aborted as they fail during validate phase while only T3 is committed. Optimistic concurrency control performs its checking at the transaction commit point in a validation phase. The serialisation order is determined by the time of the transaction validation phase.

10.7 TIMESTAMP BASED PROTOCOLS

A timestamp is used to identify the sequence of transactions. This section explains the timestamp-based concurrent execution of transactions and the multi-version technique.

10.7.1 Timestamp-Based Concurrency Control

Timestamp-based protocols use the concept of a timestamp, which is a unique value assigned to a transaction that can determine the sequence or order of transactions. The timestamp of a transaction can either be the clock time at the start of a transaction or it can be an auto-incrementing counter. In addition, the timestamping-based protocol requires that each data item be associated with the following timestamps:

1. Read timestamp: Read timestamp of a data item, say X, is the highest timestamp of the transaction that has read that data item.
2. Write timestamp: Write timestamp of a data item is the highest timestamp of the transaction that has written that data item.

Assume that a data item Di has a read timestamp rDSi and write timestamp wDSi; and a transaction with timestamp trSi makes a request to READ Di, then:

READ operation on Di:

```

IF (trSi < wDSi) then REJECT READ and ROLLBACK Ti
    //Why? The transaction is trying to read the data item, which has a
    lower timestamp than that of the highest timestamp of the transaction
    that has written the data item. Thus, this transaction is too late to read
    the data item, which is already written by a younger transaction. //
ELSE //trSi >= wDSi is TRUE//
    so ALLOW READ and
    SET rDSi = higher of (trSi, rDSi) //Set read timestamp of data item
  
```

Write operation on Di:

IF ($trSi < rDSi$) then REJECT WRITE and ROLLBACK Ti

//Why? The transaction is trying to write the data item, which is already read by a newer transaction. Therefore, this transaction is too late to write the data item.

IF ($trSi < wDSi$) then REJECT WRITE and ROLLBACK Ti

Why? The transaction is trying to write the data item, which is already written by a newer transaction. Therefore, this transaction is trying to write an old value to the data item.

ELSE // $trSi \geq rDSi$ and $trSi \geq wDSi$ //

so ALLOW WRITE and

SET $wDSi = trSi$

For example, consider the transactions T1 with timestamp 1, T2 with timestamp 2 and T3 with timestamp 3. A data item D1 is read by these transactions in the sequence T2, T1 and T3. In addition, they request to write the data item D1 in the sequence T3, T1, T2. Then how are these sequences allowed or rejected? The following Figure shows this sequence:

Request	Timestamp of the Transaction ($trSi$)	Action Performed	Read Timestamp of data item D1 ($rDS1$)	Write Timestamp of data item D1 ($wDS1$)
Initial State	-	-	0	0
READ D1 by T2	2	Allowed	2	0
READ D1 by T1	1	$trS1 < rDS1$ Reject and Rollback T1	2	0
READ D1 by T3	3	$trS3 > rDS1$ Allowed	3	0
WRITE D1 by T3	3	$trS3 \geq rDS1$ and $trS3 > wDS1$ Allowed	3	3
WRITE D1 by T1	-	This Transaction is already Rolled back	3	3
WRITE D1 by T2	2	$trS2 < rDS1$ Reject and Rollback T2	3	3

Thus, you can observe that most of the rules have been applied in the example, as given above.

10.7.2 Multi-version Technique

Multi-version technique, as the name suggests, allows the previous versions of data to be stored. Thus, this scheme avoids the rollback of transactions. This scheme also uses read timestamp and write timestamp for each version of the data item. This technique can be defined as:

Consider a data item Di and its version $Di(Ver1)$, $Di(Ver2)$, ..., $Di(VerN)$. For each $Di(Veri)$, a read timestamp $rDSi(Veri)$ and a write timestamp $wDSi(Veri)$ are stored. The read timestamp for the multi-version technique is the highest timestamp of the

transactions that have read the i^{th} version; whereas the write timestamp is the timestamp of the transaction that has resulted in the creation of this version.

In this technique read operation reads the version whose timestamp is equal to or just less than the transaction. While the case of a write operation by a transaction with timestamp trSi , which is trying to write a data item D_i that has k versions, the following three cases are possible:

- IF $\text{trSi} < \text{rDSi}(\text{Verk})$, then ROLLBACK transaction;
- IF $\text{trSi} == \text{wDSi}(\text{Verk})$, then REWRITE the version k of the data item;
- IF $\text{trSi} > \text{rDSi}(\text{Verk})$, then CREATE version $k=1$ of data item;

10.8 WEAK LEVEL OF CONSISTENCY AND SQL COMMANDS FOR TRANSACTIONS

In general, concurrent transactions are expected to be operated in serialisable mode of operation. However, in many applications, certain weaker consistency levels can also be used. These weaker consistency level puts lesser restrictions on the execution of transactions. In this section, we discuss the four different isolation levels supported by SQL, a few of which supports a weak level of consistency.

SERIALISABLE: In this isolation level, the transactions follow the ACID properties, therefore, eliminating problems due to concurrent execution of transactions, however, restricting the concurrency.

REPEATABLE READ: This isolation level allows the reading of data items only after the related transaction has been committed. However, in this isolation level a recently committed transaction may perform addition or removal of some rows.

READ COMMITTED: This isolation level also allows the reading of data items only after the related transaction has been committed. However, if a transaction repeats a read operation of a specific data item, then it is not guaranteed to obtain the same value for that data item.

READ UNCOMMITTED: This is the weakest isolation level and does not put any restriction on the concurrent execution of transactions. However, it may result in every concurrency-related problem.

The SQL command that supports transactions is:

To set a typical isolation level for transaction execution, you may set the transaction isolation level using the command:

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

You may also change the isolation level by using the alter command. You can commit a transaction by using COMMIT or roll it back by using ROLLBACK in the transaction.

You can refer to further readings for a detailed discussion on weak consistency levels and SQL commands related to transaction control.

Check Your Progress 3

- 1) Draw a suitable graph for following locking requests and find whether the transactions are deadlocked or not.

T1: S_LOCK A	--	--
--	T2: X_LOCK B	--
--	T2: S_LOCK A	--
--	--	T3: X_LOCK C
--	T2: S_LOCK C	--
T1: S_LOCK B	--	--
T1: S_LOCK A	--	--
--	--	T3: S_LOCK A

All the unlocking requests start from here

- ## 2) What is Optimistic Concurrency Control?

- 3) What are the different types of timestamps for a data item? Why are they used?

- 4) What is the purpose of the multi-version technique? Does the multi-version technique also have rollback?

- 5) What are the different weak consistency levels?

10.9 SUMMARY

In this unit you have gone through the concepts of transaction and Concurrency Management. A transaction is a sequence of many actions. Concurrency control deals with ensuring that two or more transactions do not get into each other's way, i.e., updates of data items made by one transaction do not affect the updates made by the other transactions on the same data items.

Serialisability is the generally accepted criterion for the correctness of concurrency control. It is a concept related to concurrent schedules. It determines whether any schedule is serialisable or not. Any schedule that produces the same results as a serial schedule is a serialisable schedule.

Concurrency Control is usually done via locking. If a transaction tries to lock a resource already locked by some other transaction, it cannot do so and waits for the source to be

unlocked. This unit also presents the Two-Phase Locking (2PL) protocol that ensures that transactions do not encounter concurrency-related problems. A system is in a deadlock state if there exists a set of transactions, which are waiting for each other to complete. We can use a deadlock prevention protocol to ensure that the system will never enter a deadlock state.

This unit also discusses optimistic concurrency control, timestamp-based concurrency control and multi-version technique for concurrency control. Finally, the unit discusses the different weak consistency levels and related SQL commands.

10.10 SOLUTIONS/ANSWERS

Check Your Progress 1

- 1) A transaction is the basic unit of work on a Database management system. It defines the data processing operations on the database. It has four basic properties:
 - a) Atomicity: transaction is done completely or not at all.
 - b) Consistency: Leaves the database in a consistent state
 - c) Isolation: Other transactions should not see uncommitted values
 - d) Durability: Once committed the changes should be reflected in the database.

A transaction can update more than one data values. Some transactions can do writing of data without reading a data value.

A simple transaction example may be: Updating the stock inventory of an item that has been issued.

- 2) The basic anomalies of concurrent transactions are:
 - Lost update anomaly: An update is overwritten.
 - Unrepeatable read anomaly: On reading a value later again an inconsistent value is found.
 - Dirty read anomaly: Reading an uncommitted value.
 - Inconsistent analysis: Due to reading partially updated value.

No, these problems cannot occur if the transactions do not perform interleaved read or write operations on the same data items.

- 3) The commit state arrives when a transaction has completed all the updates correctly, and all these changes are to be accepted. No, you cannot roll back after the commit.

Check Your Progress 2

- 1) There are six possible results, corresponding to six possible serial schedules:

Initially:	Sequence of operation as initially A=0	Final Value
T1-T2-T3:	A = 1; A=2; A is displayed as 2 and Set to 1	1
T1-T3-T2:	A = 1; A is displayed as 1 and set to 1; A=2	2
T2-T1-T3:	A = 0; A=1; A is displayed as 1 & set to 1	1

T2-T3-T1:	A = 0; A is displayed as 0 and set to 1; A=2	2
T3-T1-T2:	A is displayed as 0 and set to 1; A=2; A=4	4
T3-T2-T1:	A is displayed as 0 and set to 1; A=2; A=3	3

2)

Schedule	T1	T2
READ A	READ A	
A = A - 100	A = A - 100	
WRITE A	WRITE A	
READ A		READ A
READ B		READ B
READ B	READ B	
RESULT = A * B		RESULT = A * B
DISPLAY RESULT		DISPLAY RESULT
B = B + 100	B = B + 100	
WRITE B	WRITE B	

Please make the precedence graph and find out that the schedule is not serialisable.

3) This is a schedule using conservative 2 PL.

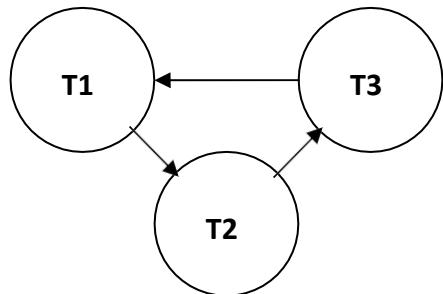
Schedule	T1	T2
Lock A	Lock A	
Lock B	Lock B	
Read A	Read A	
A = A - 100	A = A - 100	
Write A	Write A	
Unlock A	Unlock A	
Lock A		Lock A: Granted
Lock B		Lock B: Waits
Read B	Read B	
B = B + 100	B = B + 100	
Write B	Write B	
Unlock B	Unlock B	
Read A		Read A
Read B		Read B
Result = A * B		Result = A * B
Display Result		Display Result
Unlock A		Unlock A
Unlock B		Unlock B

You must also make the schedules using read and exclusive lock and a schedule in strict 2PL.

Check Your Progress 3

- 1) Transaction T1 gets the shared lock on A, T2 gets the exclusive lock on B and Shared lock on A, while transaction T3 gets the exclusive lock on C.
- Now T2 requests for a shared lock on C, which is exclusively locked by T3, so this lock cannot be granted. So T2 waits for T3 on item C.
 - T1 now requests for Shared lock on B, which is exclusively locked by T2, thus, it waits for T2 for item B. T1 request for a shared lock on C is not processed.
 - Next, T3 requests for an exclusive lock on A, which is share locked by T1, so it cannot be granted. Thus, T3 waits for T1 for item A.

The Wait for graph for the transactions for the given schedule is:



Since there exists a cycle, therefore, the schedule is deadlocked.

- 2) The basic philosophy for optimistic concurrency control is the optimism that nothing will go wrong, so let the transaction interleave in any fashion, but to avoid any concurrency-related problem, you just validate your assumption before you make changes permanent. This is a good model for situations having a low rate of transactions.
- 3) Two basic timestamps are associated with a data item – Read timestamp and Write timestamp. They are primarily used to see that a transaction that is reading data should not be too old that some younger transaction has already written it, or a transaction that wants to write that data item should not be younger than the transaction that has read or written it.
- 4) The multi-version technique of concurrent transactions ensures that all the versions or changes in the value of a data item are duly recorded. Yes, even this scheme may have a rollback.
- 5) The four weak consistency levels are - SERIALISABLE, REPEATABLE READ, READ COMMITTED, READ UNCOMMITTED.

UNIT 11 DATABASE RECOVERY AND SECURITY

- 11.0 Introduction
- 11.1 Objectives
- 11.2 What Is Recovery?
 - 11.2.1 Kinds of Failures
 - 11.2.2 Storage Structures for Recovery
 - 11.2.3 Recovery and Atomicity
 - 11.2.4 Transactions and Recovery
 - 11.2.5 Recovery in Small Databases
- 11.3 Transaction Recovery
 - 11.3.1 Log-Based Recovery
 - 11.3.2 Checkpoints in Recovery
 - 11.3.3 Recovery Algorithms
 - 11.3.4 Recovery with Concurrent Transactions
 - 11.3.5 Buffer Management
 - 11.3.6 Remote Backup Systems
- 11.4 Security in Commercial Databases
 - 11.4.1 Common Database Security Failures
 - 11.4.2 Database Security Levels
 - 11.4.3 Relationship Between Security and Integrity
 - 11.4.4 Difference Between Operating System And Database Security
- 11.5 Access Control
 - 11.5.1 Authorisation of Data Items
 - 11.5.2 A Basic Model of Database Access Control
 - 11.5.3 SQL Support for Security and Recovery
- 11.6 Audit Trails in Databases
- 11.7 Summary
- 11.8 Solutions/Answers

11.0 INTRODUCTION

In the previous unit of this block, you have gone through the details of transactions, their properties, and the management of concurrent transactions. In this unit, you will be introduced to two important issues relating to database management systems – how to deal with database failures and how to handle database security. A computer system suffers from different types of failures. A DBMS controls very critical data of an organisation and, therefore, must be reliable. However, the reliability of the database system is also linked to the reliability of the computer system on which it runs. The types of failures that the computer system is likely to be subjected to include failures of components or subsystems, software failures, power outages, accidents, unforeseen situations and natural or man-made disasters. Database recovery techniques are methods of making the database consistent till the last possible consistent state. Thus, the basic objective of the recovery system is to resume the database system to the point of failure with almost no loss of information. Further, the recovery cost should be justifiable. In this unit, we will discuss various types of failures and some of the approaches to database recovery.

The second main issue that is being discussed in this unit is Database security. “Database security” is the protection of the information contained in the database against unauthorised access, modification, or destruction. The first condition for security is to have Database integrity. “Database integrity” is the mechanism that is applied to ensure that the data in the database is consistent. In addition, the unit discusses various access

control mechanisms for database access. Finally, the unit introduces the use of audit trails in a database system.

11.1 OBJECTIVES

At the end of this unit, you should be able to:

- describe the terms recovery;
- explain log based recovery;
- explain the use of checkpoints in recovery;
- define the various levels of database security;
- define the access control mechanism;
- identify the use of audit trails in database security.

11.2 WHAT IS RECOVERY?

During the life of a transaction, i.e. after the start of a transaction but before the transaction commits, it makes several uncommitted changes in data items of a database. The database during this time may be in an inconsistent state. In practice, several things might happen to prevent a transaction from completing. Recovery techniques are designed to bring an inconsistent database, after a failure, into a consistent database state. If a transaction completes normally and commits, then all the changes made by that transaction on the database should get permanently registered in the database. They should not be lost (please recollect the durability property of transactions given in Unit 10). But if a transaction does not complete normally and terminates abnormally then all the changes made by it should be discarded.

11.2.1 Kinds of Failures

An abnormal termination of a transaction may occur due to several reasons, including:

- a) user may decide to abort the transaction issued by him/her,
- b) there might be a deadlock in the system,
- c) there might be a system failure.

The recovery mechanisms must ensure that a consistent state of the database can be restored under all circumstances. In case of transaction abort or deadlock, the system remains in control and can deal with the failure, but in case of a system failure, the system loses control because the computer itself has failed. Will the results of such failure be catastrophic? A database contains a huge amount of useful information, and any system failure should be recognised on the system restart. The DBMS should recover from any such failures. Let us first discuss the kinds of failure for ascertaining the approach of recovery.

A DBMS may encounter a failure. These failures may be of the following types:

1. Transaction failure: An ongoing transaction may fail due to:

- **Logical errors:** Transaction cannot be completed due to some internal error condition.
- **Database system errors:** A database system error can be caused by some failure at the level of a database. For example, a transaction deadlock may result in a database system error. This will result in the abrupt termination of some of the ongoing deadlocked transactions.

2. System crash: This kind of failure includes hardware/software failure of a computer system. In addition, a sudden power failure can also result in a system crash, which may result in the following:

- Loss or corruption of non-volatile storage contents.
- Loss of contents of the entire disk or parts of the disk. However, such loss is assumed to be detectable; for example, the checksums used on disk drives can detect this failure.

11.2.2 Storage Structures for Recovery

All these failures result in the inconsistent state of a database. Thus, we need a recovery scheme in a database system, but before we discuss recovery, let us briefly define the storage structure from the recovery point of view.

There are various ways for storing information for database system recovery. These are:

Volatile storage: Volatile storage does not survive system crashes. Examples of volatile storage are - the main memory or cache memory of a database server.

Non-volatile storage: The non-volatile storage survives the system crashes if it does not involve disk failure. Examples of non-volatile storage are - magnetic disk, magnetic tape, flash memory, and non-volatile (battery-backed) RAM.

Stable storage: This is a mythical form of storage structure that is assumed to survive all failures. This storage structure is assumed to maintain multiple copies on distinct non-volatile media, which may be independent disks. Further, data loss in case of disaster can be protected by keeping multiple copies of data at remote sites. In practice, software failures are more common than hardware failures. Fortunately, recovery from software failures is much quicker.

11.2.3 Recovery and Atomicity

The concept of recovery relates to the atomic nature of a transaction. Atomicity is the property of a transaction, which states that a transaction is a complete unit. Thus, the execution of a part transaction can lead to an inconsistent state of the database, which may require database recovery. Let us explain this with the help of an example:

Assume that a transaction transfers Rs.2000/- from A's account to B's account. For simplicity, we are not showing any error checking in the transaction. The transaction may be written as:

Transaction T1:

```
READ A  
A = A - 2000  
WRITE A  
→ Failure  
READ B  
B = B + 2000  
WRITE B  
COMMIT
```

What would happen if the transaction failed after account A has been written back to the database? As far as the holder of account A is concerned s/he has transferred the money but that has never been received by account holder B.

Why did this problem occur? Because although a transaction is atomic, yet it has a life cycle during which the database gets into an inconsistent state and failure has occurred at that stage.

What is the solution? In this case, where the transaction has not yet committed the

changes made by it, the partial updates need to be undone.

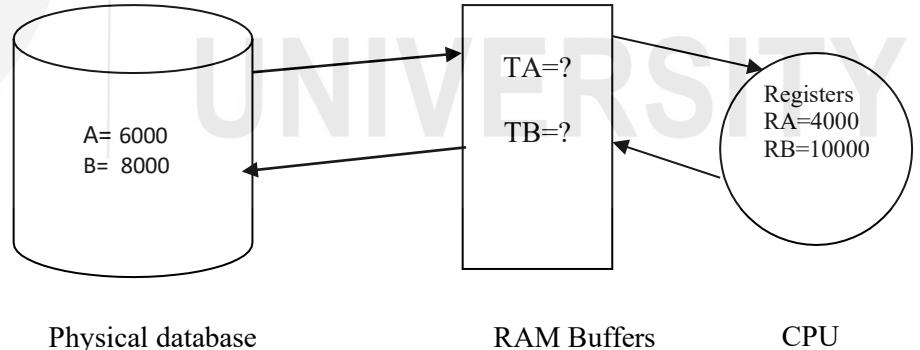
How can we do that? By remembering information about a transaction such as - when did it start, what items it updated etc. All such details are kept in a log file. We will study log files later in this unit when we define different methods of recovery. In the next section, we discuss the recovery of transactions in more detail.

11.2.4 Transactions and Recovery

The basic unit of recovery is a transaction. But how are the transactions handled during recovery?

Consider the following two cases:

- i. Consider that some transactions are deadlocked, then at least one of these transactions must be restarted to break the deadlock, and thus, the partial updates made by this restarted transaction program are to be undone to keep the database in a consistent state. In other words, you may ROLLBACK the effect of a transaction.
- ii. A transaction has committed, but the changes made by the transaction have not been communicated to the physical database on the hard disk. A software failure now occurs, and the contents of the CPU/ RAM are lost. This leaves the database in an inconsistent state. Such failure requires that on restarting the system the database be brought to a consistent state using **redo** operation. The redo operation performs the changes made by the transaction again to bring the system to a consistent state. The database system can then be made available to the users. The point to be noted here is that such a situation has occurred as database updates are performed in the buffer memory. *Figure 1* shows cases of **undo** and **redo**.



	Physical Database at the time of failure	RAM	Activity
Case 1	A=6000 B=8000	TA=4000 TB=8000	Transaction T1 has changed the value of A in register RA (4000) and RB (10000). RA is written to RAM (TA), but not updated in A. RB is not written back to RAM (TB). T1 did not COMMIT. Now, T1 is aborted by the user. The value of CPU registers and RAM

			are made irrelevant. You cannot determine if the TA, TB has been written back to A, B. You must UNDO the transaction.
Case 2	A=4000 B=8000	TA=4000 TB=8000	The value of A in the physical database has got updated due to buffer management. RB is not written back to RAM (TB). The transaction did not COMMIT so far. Now, the transaction T1 aborts. You must UNDO the transaction.
Case 3	A=6000 B=8000	TA=4000 TB=10000 COMMIT	The value B in the physical database has not got updated due to buffer management. T1 has raised the COMMIT flag. The changes of the transaction must be performed again. You must REDO the transaction. How? (Discussed in later sections).

Figure 1: Database Updates and Recovery

11.2.5 Recovery in Small Databases

Failures can be handled using different recovery techniques that are discussed later in the unit. But the first question is: Do you really need recovery techniques as a failure control mechanism? The recovery techniques are somewhat expensive both in terms of time and memory space for small systems. In such a case, it is beneficial to avoid failures by some checks instead of deploying recovery techniques to make the database consistent. Also, recovery from failure involves manpower that can be used in other productive work if failures can be avoided. It is, therefore, important to find some general precautions that help control failures. Some of these precautions may be:

- to regulate the power supply.
- to use a failsafe secondary storage system such as RAID.
- to take periodic database backups and keep track of transactions after each recorded state.
- to properly test transaction programs prior to use.
- to set important integrity checks in the databases as well as user interfaces.

However, it may be noted that if the database system is critical to an organisation, it must use a DBMS that is suitably equipped with recovery procedures.

11.3 TRANSACTION RECOVERY

As discussed in the previous section, a transaction is the unit of recovery. A commercial database system, like a banking system, may support many concurrent transactions at a time. A failure may affect multiple transactions in such systems. Several recovery techniques have been designed for commercial DBMSs. This section discusses some of the basic recovery schemes used in commercial DBMSs.

11.3.1 Log-Based Recovery

Let us first define the term transaction log in the context of DBMS. A transaction log, in DBMS, records information about every transaction that modifies any data values in the database. A log contains the following information about a transaction:

- A transaction BEGIN marker.
- Transaction identification - transaction ID, terminal ID, user ID, etc.
- The operations being performed by the transaction such as UPDATE, DELETE, INSERT.
- The data items or objects affected by the transaction - may include the table's name, row number and column number.
- Before or previous values (also called UNDO values) and after or changed values (also called REDO values) of the data items that have been updated.
- A pointer to the next transaction log record, if needed.
- The COMMIT marker of the transaction.

In a database system, several transactions run concurrently. When a transaction commits, the data buffers used by it need not be written back to the physical database stored on the secondary storage as these buffers may be used by several other transactions that have not yet committed. On the other hand, some of the data buffers that may have been updated by several uncommitted transactions might be forced back to the physical database, as they are no longer being used by the database. So, the transaction log helps in remembering which transaction did what changes. Thus, the system knows exactly how to separate the changes made by transactions that have already committed from those changes that are made by the transactions that did not yet COMMIT. Any operation such as BEGIN transaction, INSERT/ DELETE/ UPDATE and transaction COMMIT, adds information to the log containing the transaction identifier and enough information to UNDO or REDO the changes.

But how do we recover using a log? Let us demonstrate this with the help of an example having three concurrent transactions that are active on ACCOUNTS relation:

Transaction T1	Transaction T2	Transaction T3
READ X	READ A	READ Z
SUBTRACT 100	ADD 200	SUBTRACT 500
WRITE X	WRITE A	WRITE Z
READ Y		
ADD 100		
WRITE Y		

Figure 2: The sample transactions

Assume that these transactions have the following log file (hypothetical structure):

Transaction Begin Marker	Transaction Id	Operation on ACCOUNTS table	UNDO values (assumed)	REDO values	Transaction Commit Marker
Yes	T1	SUB ON X ADD ON Y	X=500 Y=800	X=400 not done yet	No
Yes	T2	ADD ON A	A=1000	A=1200	No
Yes	T3	SUB ON Z	Z=900	Z=400	Yes

Figure 3: A sample (hypothetical) Transaction log

Now assume at this point of time a failure occurs, then how the recovery of the database will be done on restart.

Transaction /Values	Initial (UNDO value)	Just before the failure	Operation Required for recovery	Database Values after Recovery
T1/X	500	400 (assuming update has been done in physical database also)	UNDO (As transaction did not COMMIT)	X = 500
T1/Y	800	800	UNDO	Y = 800

T2/A	1000	1000 (assuming update has not been done in physical database)	UNDO (As transaction did not COMMIT)	A = 1000
T3/Z	900	900 (assuming update has not been done in physical database)	REDO (As transaction did COMMIT)	Z = 400

Figure 4: The database recovery

The selection of REDO or UNDO for a transaction for the recovery is done based on the state of the transactions. This state is determined in two steps:

- Look into the log file and find all the transactions that have started. For example, in *Figure 3*, transactions T1, T2 and T3 are candidates for recovery.
- Find those transactions that have COMMITTED. REDO these transactions. All other transactions have not COMMITTED, so they should be rolled back, so UNDO them. For example, in *Figure 3*, UNDO will be performed on T1 and T2, and REDO will be performed on T3.

Please note that in Figure 4, some of the values may not have yet been communicated to the database, yet we need to perform UNDO as we are not sure what values have been written back to the database. Similarly, you must perform REDO operations on committed transactions, such as Transaction T3 in Figure 3 and Figure 4.

But how will the system recover? Once the recovery operation has been specified, the system just determines the required REDO or UNDO values from the transaction log and changes the inconsistent state of the database to a consistent state. (Please refer to *Figure 3* and *Figure 4*).

13.3.2 Checkpoints in Recovery

Let us consider several transactions, which are shown on a timeline, with their respective START and COMMIT time (see Figure 5).

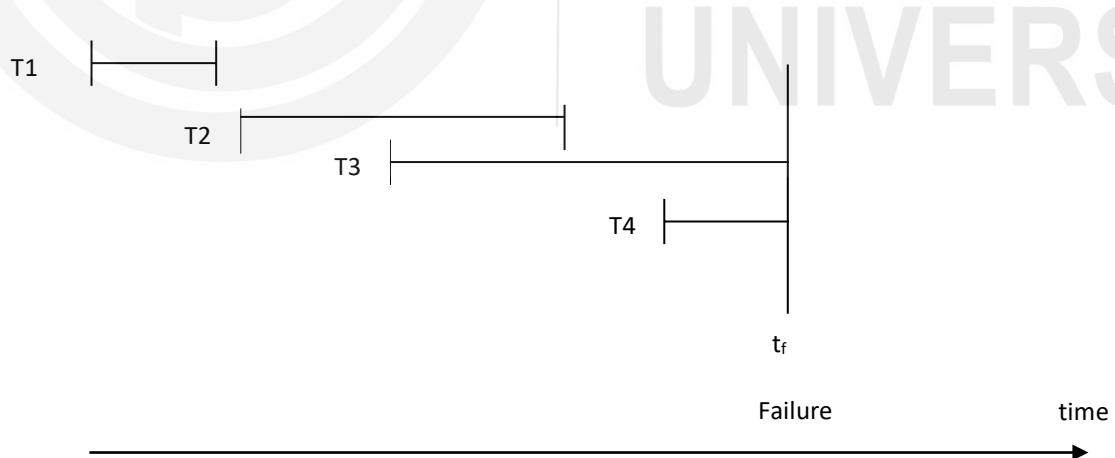


Figure 5: Execution of Concurrent Transactions

Consider that the transactions are allowed to execute concurrently, on encountering a failure at time t_f , the transactions T1 and T2 are to be REDONE and T3 and T4 will be UNDONE (Refer to Figure 5). Now, considering that a system allows thousands of parallel transactions, then all those transactions that have been committed may have to be redone, and all uncommitted transactions need to be undone. That is not a very good

choice as it requires redoing even those transactions that might have been committed several hours earlier. How can you improve this situation? You can remedy this problem by taking a checkpoint. *Figure 6* shows a checkpoint mechanism:

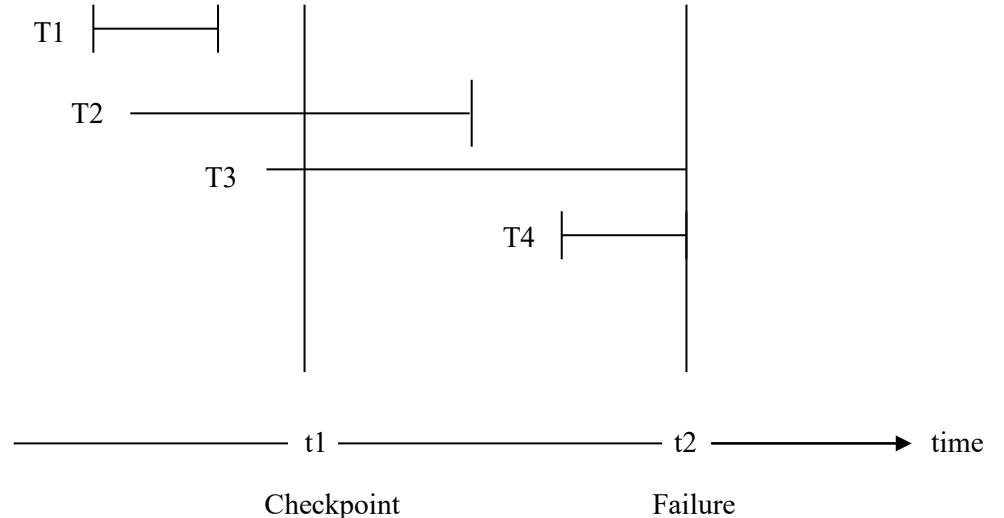


Figure 6: Checkpoint in Transaction Execution

A checkpoint is taken at time t_1 , and a failure occurs at time t_2 . Checkpoint transfers all the committed changes to the database and all the system logs to stable storage (the storage that would not be lost). On the restart of the system after the failure, the stable checkpointed state is restored. Thus, we need to REDO or UNDO only those transactions that have been completed or started after the checkpoint has been taken. A disadvantage of this scheme is that the database would not be available when the checkpoint is being taken. In addition, some of the uncommitted data values may be put in the physical database. To overcome the first problem, the checkpoints should be taken at times when the system load is low. To avoid the second problem, the system may allow the ongoing transactions to be complete while not starting any new transactions.

In the case of *Figure 6*, the recovery from failure at time t_2 will be as follows:

- The transaction T1 will not be considered for recovery, as the changes made by it have already been committed and transferred to the physical database at checkpoint t_1 .
- The transaction T2 has not committed till checkpoint t_1 but has committed before t_2 will be REDONE.
- T3 must be UNDONE as the changes made by it before the checkpoint (we do not know for sure if any such changes were made prior to the checkpoint) must have been communicated to the physical database. T3 must be restarted with a new name.
- T4 started after the checkpoint, and if we strictly follow the scheme in which the buffers are written back only on the checkpoint, then nothing needs to be done except restart the transaction T4 with a new name.

The restart of a transaction requires the log to keep information on the new name of the transaction. This new transaction may be given higher priority.

But one question that remains unanswered is - during a failure, we lose database information in RAM buffers; we may also lose the content of the log as it is also stored in RAM buffers, so how does the log ensure recovery?

The answer to this question lies in the fact that for storing the log of the transaction, we follow a **Write Ahead Log Protocol**. As per this protocol, the transaction logs are written to stable storage as follows:

- **UNDO portion of the log is written to stable storage prior to any updates and**
- **REDO portion of the log is written to stable storage prior to the commit.**

Log-based recovery scheme can be used for any kind of failure provided you have stored the most recent checkpoint state and most recent log as per write-ahead log protocol into the stable storage. Stable storage from the viewpoint of external failure requires more than one copy of such data at more than one location. You can refer to further readings for more details on recovery and its techniques.

☛ Check Your Progress 1

1) What is the need for recovery? What is the basic unit of recovery?

.....
.....

2) What is the log-based recovery?

.....
.....

3) What is a checkpoint? Why is it needed? How does a checkpoint help in recovery?

.....
.....

11.3.3 Recovery Algorithms

As discussed earlier, a database failure may bring the database into an inconsistent state, as many ongoing transactions will simply abort. In order to bring such an inconsistent database to a consistent state, you are required to use recovery algorithms. Database recovery algorithms require basic data related to failed transactions. Thus, a recovery algorithm requires the following actions:

- 1) Actions are taken to collect the required information for recovery while the transactions are being processed prior to the failure.
- 2) Actions are taken after a failure to recover the database contents to a state that ensures atomicity, consistency and durability.

In the context of point 1 above, it may be noted that the information about the changes made by a transaction is recorded in a log file. In general, the sequence of logging during the transaction execution is as follows:

- A transaction, say T_i , announces its start by putting a log record consisting of $\langle T_i \text{ start} \rangle$.
- Before T_i executes the **write**(X) (see Figure 7), put a log record $\langle T_i, X, V_1, V_2 \rangle$, where V_1 represented the older value of X , i.e., the value of X before the write (*undo* value), and V_2 is the updated value of X (*redo* value).
- On successful completion of the last statement of the transaction T_i , a log

record consisting of $\langle T_i \text{ commit} \rangle$ is put in the log. It may be noted that all these log records are put in stable storage media, that is, they are not buffered in the main memory).

Two approaches for recovery using logs are:

- Deferred database modification.
- Immediate database modification.

Deferred Database Modification

This scheme logs all the changes made by a transaction into a log file, which is stored on stable storage. Further, these changes are not made in the database till the transaction commits. Let us assume that transactions execute serially to simplify the discussion. Consider a transaction T_i , it performs the following sequence of log file actions:

Event	Record for Log file	Comments
Start of Transaction	$\langle T_i \text{ start} \rangle$	
write (X)	$\langle T_i, X, V \rangle$	Transaction writes a new value V for data item X in the database.
Transaction T_i commits	$\langle T_i \text{ commit} \rangle$	The transaction has been committed, and now the deferred updates can be written to the database

- Please note that in this approach, the old values of the data items are not saved at all, as changes are being recorded in the log file and the database is not changed till a transaction commits. For example, the **write**(X), shown above, does not write the value of X in the database till the transaction commits. However, the record $\langle T_i, X, V \rangle$ is written to the log. All these updates are written to the database after the transaction commits.

How is the recovery performed for the deferred database modification scheme?

For the database recovery after the failure, the log file is checked. The transactions for which the log file contains the $\langle T_i \text{ start} \rangle$ and the $\langle T_i \text{ commit} \rangle$ records, the REDO operation is performed using log record $\langle T_i, X, V \rangle$. Why? Because in the deferred update scheme we do not know if the changes made by this committed transaction have been carried out in the physical database or not. The Redo operation can be performed many times without the loss of information, so it will be applicable even if the crash occurs during recovery. The transactions for which $\langle T_i \text{ start} \rangle$ is in the log file but not the $\langle T_i \text{ commit} \rangle$, the transaction UNDO is not required, as it is expected that the values are not yet written to the database.

Let us explain this scheme with the help of the transactions given in Figure 7.

$T_1: \text{READ } (X)$
 $X = X - 1000$
WRITE (X)
READ (Y)
 $Y = Y + 1000$
WRITE (Y)

$T_2: \text{READ } (Z)$
 $Z = Z - 1000$
WRITE (Z)

Sample Transactions T_1 and T_2 (T_1 executes before T_2)
Figure 7: Sample Transaction for demonstrating Recovery Algorithm

Figure 8 shows the state of a sample log file for three possible failure instances, namely

(a), (b) and (c). (Assuming that the initial balance in X is 10,000/- Y is 5,000/- and Z has 20,000/-):

<T ₁ START>	<T ₁ START >	<T ₁ START>
<T ₁ , X, 9000>	<T ₁ , X, 9000>	<T ₁ , X, 9000>
<T ₁ , Y, 6000>	<T ₁ , Y, 6000>	<T ₁ , Y, 6000>
	<T ₁ COMMIT>	<T ₁ COMMIT >
	<T ₂ START >	<T ₂ START >
	<T ₂ , Z, 19000>	<T ₂ , Z, 19000>
(a)	(b)	(c)

Figure 8: Log Records for Deferred Database Modification for Transactions of Figure 7

Why do you store only the Redo value in this scheme? The reason is that No UNDO is required as updates are communicated to stable storage only after COMMIT or even later. The following REDO operations would be required if the log on stable storage at the time of the crash is as shown in Figure 8(a) 8(b) and 8(c).

- (a) No REDO is needed, as no transaction has been committed in Figure 8(a).
- (b) Perform REDO(T_1), as $\langle T_1 \text{ COMMIT} \rangle$ is part of the log in Figure 8 (b).
- (c) Perform REDO(T_1) and REDO(T_2), as $\langle T_1 \text{ COMMIT} \rangle$ and $\langle T_2 \text{ COMMIT} \rangle$ are in the log shown in Figure 8(c).

Please note that you can repeat this sequence of redo operation as suggested in Figure 8(c) any number of times, it will still bring the value of X, Y, and Z to consistent redo values. This property of the redo operation is called **idempotent**.

Immediate Database Modification

This recovery scheme allows the modification of the data items of the stored database by ongoing uncommitted transactions. Thus, the recovery database would require UNDO and REDO of updates made by these transactions. Therefore, the log file for this recovery scheme should include the UNDO value or the original value and the REDO value or the modified value. Further, the log records are assumed to be written to the stable storage.

The following log file shows the log of transactions given in Figure 7, for the case when immediate database modification is followed:

Log	Write operation	Output
< T_1 START>		
< T_1 , X, 10000, 9000>	$X = 9000$	Output Block of X
< T_1 , Y, 5000, 6000>	$Y = 6000$	Output Block of Y
< T_1 COMMIT>		
< T_2 START>		
< T_2 , Z, 20,000, 19,000>	$Z = 19000$	Output Block of Z
< T_2 COMMIT>		

Figure 9: Log Records for Immediate Database Modification for Transactions of Figure 7

The UNDO and REDO operations for this recovery scheme are as follows:

- **UNDO(T_i):**
 - List all the transactions which have the start record in the log but no commit record. For all the uncommitted transactions in the list perform the following:
 - Find the last UNDO log record of a transaction.
 - Move backwards in the log file to find all the entries for UNDO of

- that transaction.
- For each UNDO entry, assign the UNDO value or original value to the data item value.
- REDO(T_i):**
 - List all the transactions which have the start record and commit record in the log. For all the committed transactions in the list, perform the following:
 - Find the first REDO log record of a transaction.
 - Move forward in the log file to find all the entries for REDO of that transaction.
 - For each REDO entry, assign the REDO value or modified value to the data item value.

You may note that you may perform REDO/UNDO operations any number of times if needed. Further, UNDO operations are performed first, followed by the REDO operations.

Example:

Consider the log as it appears at three instances of time.

$\langle T_1 \text{ start} \rangle$ $\langle T_1, X 10000, 9000 \rangle$ $\langle T_1, Y 5000, 6000 \rangle$ Failure (a)	$\langle T_1 \text{ start} \rangle$ $\langle T_1, X 10000, 9000 \rangle$ $\langle T_1, Y 5000, 6000 \rangle$ $\langle T_1, \text{Commit} \rangle$ $\langle T_2 \text{ start} \rangle$ $\langle T_2, Z 20000, 19000 \rangle$ Failure (b)	$\langle T_1 \text{ start} \rangle$ $\langle T_1, X 10000, 9000 \rangle$ $\langle T_1, Y 5000, 6000 \rangle$ $\langle T_1, \text{Commit} \rangle$ $\langle T_2 \text{ start} \rangle$ $\langle T_2, Z 20000, 19000 \rangle$ $\langle T_2, \text{Commit} \rangle$ Failure (c)
--	--	--

Figure 10: Recovery in Immediate Database Modification technique

For each of the failures as shown in Figure 10 (a), (b) and (c), the following recovery actions would be needed:

- (a) UNDO (T_1):
 - $X \leftarrow$ Undo value of X, i.e. 10000;
 - $Y \leftarrow$ Undo value of Y, i.e. 5000.
- (b) UNDO (T_2):
 - $Z \leftarrow$ Undo value of Z, i.e. 20000;
- REDO (T_1):
 - $X \leftarrow$ Redo value of X, i.e. 9000;
 - $Y \leftarrow$ Redo value of Y, i.e. 6000.
- (c) REDO (T_1, T_2) by moving forward:
 - $X \leftarrow$ Redo value of X, i.e. 9000;
 - $Y \leftarrow$ Redo value of Y, i.e. 6000;
 - $Z \leftarrow$ Redo value of Z, i.e. 19000;

Advanced Recovery Techniques

The recovery processes for DBMS have gone through several changes. One of the basic objectives of any recovery technique is to minimise the time that is required to perform the recovery. However, in general, this reduction in time results in increased complexity of the recovery algorithms. One of the recent popular log-based recovery techniques is named ARIES. You may refer to further reading for more details on ARIES. In general, several aspects that increase the speed of the recovery process are:

- Transaction log sequence numbers for log records may be assigned to log entries. This will help in identifying the related database log pages.
- Instead of deletion the records from the physical database, record the deletion in the log.
- Keep track of pages that have been updated in memory but have not been written back to the physical database. Perform Redo operations for only such pages. Also, keep track of updated pages during checkpointing.

You may refer to further readings for more details on newer methods and algorithms of recovery.

11.3.4 Recovery with Concurrent Transactions

In general, a commercial database management system, such as a banking system, has many users. These users can perform multiple transactions. Therefore, a centralised database system executes many concurrent transactions. As discussed in Unit 10, these transactions may experience concurrency-related issues and, thus, are required to maintain serialisability. In general, these transactions share a large buffer area and log files. Thus, a recovery scheme that allows better control of disk buffers and large log files should be employed for such systems. One such technique is called checkpointing. This changes the extent to which REDO or UNDO operations are to be performed in a database system. The concept of the checkpoint has already been discussed in section 11.3.2. How checkpoints can help in the process of recovery is explained below:

When you use the checkpoint mechanism, you also add records of checkpoints in the log file. A checkpoint record is of the form: <checkpoint TL>. In this case, TL is the list of transactions, which were started, but not yet committed at the time when the checkpoint was created. Further, we assume that at the time of creating a checkpoint record, no transaction was allowed to proceed.

On the restart of the database system after failure, the following steps are performed for the database recovery:

- Create two lists: UNDOLIST and REDOLIST. Initialise both lists to a NULL.
- Scan the log file for every log record, starting from the end of the file and moving backwards, till you locate the first checkpoint record <checkpoint TL>. Perform the following actions, for each of the log records found in this step:
 - Is the log record < T_i COMMIT>?
 - If yes, then add T_i to REDOLIST.
 - Is the log record < T_i START>?
 - If yes, then Is T_i NOT IN REDOLIST?
 - Add T_i to UNDOLIST, as it has not yet been committed.
 - For every T_i in TL : Is T_i in NOT IN REDOLIST?
 - add T_i to UNDOLIST, as this transaction was active at the time of checkpoint and has not been committed yet.

This will make the UNDOLIST and REDOLIST of the transactions. Now, you can perform the UNDO operations followed by REDO operations using the log file, as given in section 11.3.3.

11.3.5 Buffer Management

As discussed in the previous sections, the database transactions are executed in the memory, which contains a copy of the physical database. These database buffers are written back to stable storage from time to time. In general, it is the memory management service of the operating system that manages the buffers of a database system. However, several database management schemes require their own buffer management policies and, hence, the buffer management system. Some of these strategies are:

Log Record Buffering

The recovery process requires database transactions to write logs in stable storage. This is a very time-consuming process, as for a single transaction several log records are to be written to stable storage. Therefore, several commercial database management systems perform the buffering of the log file itself, which means that log records are kept in the blocks of the main memory allocated for this purpose. The logs are then written to the stable storage once the buffer becomes full or when a transaction commits. This log file buffering helps in reducing disk accesses, which are very expensive in terms of time of operation.

Database recovery requires that log records should be stored in stable storage. Therefore, log records should be transferred from memory buffers to the table storage as per the following scheme, called Write-Ahead logging.

- The sequence of log records in the memory buffer should be maintained in stable storage.
- A transaction should be moved to COMMIT state only if the $\langle T_i \text{ commit} \rangle$ log record is written to stable storage.
- Prior to writing a database buffer to stable storage, related log records in the buffer should be moved to stable storage.

Database Buffering

The database updates are performed after moving database blocks on secondary storage to memory buffers. However, due to limited memory capacity, only a few database blocks can be kept in the memory buffers. Therefore, database buffer management consists of policies for deciding which blocks should be kept in database buffers and what blocks should be removed from the database buffers back to secondary storage. Removing a database block from the buffer requires that it is re-written to the secondary storage. In addition, the log is written to stable storage, as per the write-ahead logging.

11.3.6 Remote Backup Systems

Most of the techniques discussed above perform recovery in the context of a centralised system. However, present-day transaction processing systems require high availability. One of the ways of implementing a high availability system is to create a primary and a backup site for the transaction processing system. With fast, highly reliable networks, this backup site can be a remote backup site. This remote backup site may be very useful in case of disaster recovery. Some of the issues of the remote backup system include the following:

- The failure of the primary database site must be detected. It may be noted that this detection should not detect communication failure as a primary database failure. Thus, it may use a failsafe communication, which may have alternative communication links to the primary database site.
- The backup site should be capable enough to work as the primary database site at the time of failure of the primary site. In addition to recovery of ongoing transactions, once the primary site recovers it should get all the updates, which were performed while the primary site was down.

You may refer to the further readings for more details on this topic.

☞ Check Your Progress 2

- 1) What is deferred database modification in recovery algorithms?

.....

2) How is log of the deferred database modification technique differ from the log of Immediate database modification?

.....

.....

3) Define buffer management and remote backup system in the context of recovery.

.....

.....

11.4 SECURITY IN COMMERCIAL DATABASES

You must realise that security is a journey, not the final destination. You cannot assume a product/technique is absolutely secure, as you may not be aware of fresh/new attacks on that product/technique. Many security vulnerabilities are not even published as attackers want to delay a fix, and manufacturers do not want negative publicity. There is an ongoing and unresolved discussion over whether highlighting security vulnerabilities in the public domain encourages or prevents further attacks.

The most secure database you can think of must be found in a most securely locked bank or nuclear-proof bunker, installed on a standalone computer without an Internet or network connection, and under guard for $24 \times 7 \times 365$. However, that is not a likely scenario with which we would like to work. A database server maintains database services, which often contain security issues, and you should be realistic about possible threats. You must assume security failure at some point and never store truly sensitive data in a database that unauthorised users may easily infiltrate/access. A major point here is that most data loss occurs because of social exploits and not technical ones. Thus, the use of encryption algorithms for security may need to be looked into.

You will be able to develop effective database security if you realise that securing data is essential to the market reputation, profitability and business objectives. For example, personal information such as credit cards or bank account numbers are now commonly available in many databases; therefore, there are more opportunities for identity theft. As per an estimate, several identity theft cases are committed by employees who have access to large financial databases. Banks and companies that take credit card services externally must place greater emphasis on safeguarding and controlling access to this proprietary database information.

Securing the database is a fundamental tenet for any security personnel while developing his or her security plan. The database is a collection of useful data and can be treated as the most essential component of an organisation and its economic growth. Therefore, for any security effort, you must keep in mind that you need to provide the strongest level of control over the data of a database.

As is true for any other technology, the security of database management systems depends on many other systems. These primarily include the operating system, the applications that use the DBMS, services that interact with the DBMS, the web server that makes the application available to end users, etc. However, please note that most importantly, DBMS security depends on us, the users.

11.4.1 Common Database Security Failures

Database security is of paramount importance for an organisation, but many organisations do not take this fact into consideration till an eventual problem occurs. The common pitfalls that threaten database security are:

Weak User Account Settings: Many of the database user accounts do not contain the user settings that may be found in operating system environments. For example, the user accounts name and passwords, which are commonly known, are not disabled or modified to prevent access.

Insufficient Segregation of Duties: Several organisations have no established security administrator role. This results in database administrators (DBAs) performing both the functions of the administrator (for users' accounts), as well as the performance and operations expert. This may result in management inefficiencies.

Inadequate Audit Trails: The auditing capabilities of DBMS, since it requires keeping track of additional requirements, are often ignored for enhanced performance or disk space. Inadequate auditing results in reduced accountability. It also reduces the effectiveness of data history analysis. The audit trails record information about the actions taken on certain critical data. They log events directly associated with the data; thus, they are essential for monitoring the access and the activities on a database system.

Unused DBMS Security Features: The security of an individual application is usually independent of the security of the DBMS. Please note that security measures that are built into an application apply to users of the client software only. The DBMS itself and many other tools or utilities that can connect to the database directly through ODBC or any other protocol may bypass this application-level security completely. Thus, you must try to use security restrictions that are reliable, for instance, try using the security mechanisms that are defined within the DBMS.

11.4.2 Database Security Levels

Basically, database security can be broken down into the following levels:

- Server Security
- Database Connections
- Table Access Control

Server Security: Server security is the process of controlling access to the database server. This is the most important aspect of security and should be carefully planned. The basic idea here is “You cannot access what you do not see”. For security purposes, you should never let your database server be visible to the world. If a database server is supplying information to a web server, then it should be configured in such a manner that it is allowed connections from that web server only. Such a connection would require a trusted IP address.

Trusted IP Addresses: To connect to a server through a client machine, you would need to configure the server to allow access to only trusted IP addresses. You should know exactly who should be allowed to access your database server. For example, if the database server is the backend of a local application that is running on the internal network, then it should only talk to addresses from within the internal network.

Database Connections: With the ever-increasing number of Dynamic Applications, an application may allow immediate unauthenticated updates to some databases. If you are going to allow users to make updates to a database via a web page, please ensure that you validate all such updates. This will ensure that all updates are desirable and safe. For example, you may remove any possible SQL code from user-supplied input if a normal user is not allowed to input SQL code.

Table Access Control: Table access control is probably one of the most overlooked but one of the very strong forms of database security because of the difficulty in applying it. Using a table access control properly would require the collaboration of both the system administrator as well as the database developer. In practice, however, such “collaboration” is relatively difficult to find.

By now, we have defined some of the basic issues of database security, let us now consider specifics of server security from the point of view of network access of the system. Internet-based databases have been the most recent targets of security attacks.

All web-enabled applications listen to a number of ports. Cyber criminals often perform a simple “port scan” to look for ports that are open from the popular default ports used by database systems. How can we address this problem? We can address this problem “by default”, that is, we can change the default ports a database service would listen into. Thus, this is a very simple way to protect the DBMS from such criminals.

11.4.3 Relationship between Security and Integrity

Database security usually refers to the avoidance of unauthorised access and modification of data of the database, whereas database integrity refers to the avoidance of accidental loss of consistency of data. You may please note that data security deals not only with data modification but also access to the data, whereas data integrity, which is normally implemented with the help of constraints, essentially deals with data modifications. Thus, enforcement of data security, in a way, starts with data integrity. For example, any modification of data, whether unauthorised or authorised must ensure data integrity constraints. Thus, a very basic level of security may begin with data integrity but will require many more data controls. For example, SQL WRITE and UPDATE on specific data items or tables would be possible if it does not violate integrity constraints. Further, the data controls would allow only authorised WRITE and UPDATE on these data items.

11.4.4 Difference between Operating System and Database Security

Security within the operating system can be implemented at several levels ranging from passwords for access to the operating system to the isolation of concurrently executing processes within the operating system. However, there are a few differences between security measures taken at the operating system level compared to those of database system. These are:

- Database system protects more objects, as the data is persistent in nature. Also, database security is concerned with different levels of granularity such as files, tuples, attribute values or indexes. Operating system security is primarily concerned with the management and use of resources.
- Database system objects can be complex logical structures such as views, a number of which can map to the same physical data objects. Moreover, different architectural levels viz. internal, conceptual and external levels, have different security requirements. Thus, database security is concerned with the semantics – meaning of data, as well as with its physical representation. The operating system can provide security by not allowing any operation to be performed on the database unless the user is authorised for the operation concerned.

After this brief introduction to different aspects of database security, let us discuss one of the important levels of database security, access control, in the next section.

11.5 ACCESS CONTROL

All relational database management systems provide some sort of intrinsic security mechanisms that are designed to minimise security threats, as stated in the previous sections. These mechanisms range from the simple password protection offered in Microsoft Access to the complex user/role structure supported by advanced relational databases like Oracle, MySQL, Microsoft SQL Server, IBM Db2 etc. But can we define access control for all these DBMS using a single mechanism? SQL provides that interface for access control. Let us discuss the security mechanisms common to all databases using the Structured Query Language (SQL).

An excellent practice is to create individual user accounts for each database user. If

users are allowed to share accounts, then it becomes very difficult to fix individual responsibilities. Thus, it is important that we provide separate user accounts for separate users. Does this mechanism have any drawbacks? If the expected number of database users is small, then it is all right to give them individual usernames and passwords and all the database access privileges that they need to have on the database items.

However, consider a situation where there are a large number of users. Specification of access rights to all these users individually will take a long time. That is still manageable as it may be a one-time effort; however, the problem will be compounded if we need to change the access rights for a particular user. Such an activity would require huge maintenance costs. This cost can be minimised if we use a specific concept called “Roles”. A database may have hundreds of users, but their access rights may be categorised in specific roles, for example, teacher and student in a university database. Such roles would require the specification of access rights only once for each **role**. The users can then be assigned usernames, passwords, and specific roles. Thus, the maintenance of user accounts becomes easier as now we have limited roles to be maintained. You may study these mechanisms in the context of specific DBMS. A role can be defined using a set of data item/object authorisations. In the next sections, we define some of the authorisations in the context of SQL.

11.5.1 Authorisation of Data Items

Authorisation is a set of rules that can be used to determine which user has what type of access to which portion of the database. The following forms of authorisation are permitted on database items:

- 1) **READ:** it allows reading of data objects, but not modification, deletion, or insertion of a data object.
- 2) **INSERT:** allows insertion of new data, for example, insertion of a tuple in a relation, but it does not allow the modification of existing data.
- 3) **UPDATE:** allows modification of data, but not its deletion. However, data items like primary-key attributes may not be modified.
- 4) **DELETE:** allows deletion of data only.

A user may be assigned all, none, or a combination of these types of authorisations, which are broadly called access authorisations.

In addition to these manipulation operations, a user may be granted control operations such as:

- 1) **ADD:** allows adding new objects such as new relations.
- 2) **DROP:** allows the deletion of relations in a database.
- 3) **ALTER:** allows the addition of new attributes in a relation or deletion of existing attributes in a relation.
- 4) **Propagate Access Control:** this is an additional right that allows a user to propagate the access control or access right which s/he already has to some other user, for example, if user A has access right R over a relation S and if s/he has right to propagate access control, then s/he can propagate her/his access right R over relation S to another user B either fully or partially. In SQL, you can use WITH GRANT OPTION for this right.

The ultimate form of authority is given to the database administrator. S/he is the one who may authorise new users, restructure the database and so on. The process of authorisation involves supplying information only to the person who is authorised to access that information.

11.5.2 A basic model of Database Access Control

Models of database access control have grown out of earlier work on protection in

operating systems. Let us discuss one simple model with the help of the following example:

Example

Consider the relation:

Employee (Empno, Name, Address, Deptno, Salary, Assessment)

Assume there are two types of users: The personnel manager and the general user. What access rights may be granted to each user? One extreme possibility is to grant unconstrained access or to have limited access. One of the most influential protection models was developed by Lampson and extended by Graham and Denning. This model has 3 components:

- 1) A set of object entities to which access must be controlled.
- 2) A set of subject entities that request access to objects.
- 3) A set of access rules in the form of an authorisation matrix, as given in Figure 11 for the relation of the example.

Object \ Subject	Empno	Name	Address	Deptno	Salary	Assessment
Personnel Manager	Read	Read	All	All	All	All
General User	Read	Read	Read	Read	Not accessible	Not accessible

Figure 11: Authorisation Matrix for Employee relation.

As the above matrix shows, Personnel Manager and General User are the two subjects. Objects of the database are Empno, Name, Address, Deptno, Salary and Assessment. As per the access matrix, the personnel manager can perform any operation on the database of an employee except for updating the Empno and Name, which may be created once and can never be changed. The general user can only read the data but cannot update, delete or insert the data into the database. Also, the information about the salary and assessment of the employee is not accessible to the general user.

In summary, it can be said that the basic access matrix is the representation of basic access rules. These rules can be written using SQL statements, which are given in the next subsection.

11.5.3 SQL Support for Security and Recovery

You would need to create the users or roles before you grant them various permissions. The permissions then can be granted to a created user or role. This can be done with the use of the SQL GRANT statement.

The syntax of this statement is:

```
GRANT <permissions> [ON <table/view>] TO <user/role>
[WITH GRANT OPTION]
```

Let us define this statement line-by-line. The first line, GRANT <permissions>, allows you to specify the specific permissions on a table or a database view. These can be either relation-level data manipulation permissions (such as SELECT, INSERT, UPDATE and DELETE) or data definition permissions (such as CREATE TABLE, ALTER DATABASE and GRANT). More than one permission can be granted in a single GRANT statement, but data manipulation permissions and data definition permissions may not be combined in a single statement.

The second line, ON <table/view>, is used to specify the table or a view on which permissions are being given. This line is not needed if we are granting data definition permissions.

The third line specifies the user(s) or role(s) that is/are being granted permissions.

Finally, the fourth line, WITH GRANT OPTION, is optional. If this line is included *in the statement*, the user is also permitted to grant the same permissions that s/he has received to other users. Please note that the WITH GRANT OPTION cannot be specified when permissions are assigned to a *role*.

Let us look at a few examples of the use of this statement.

Example 1: Assume that you have recently hired a group of 25 data entry operators who will be adding and maintaining student records in a University database system. They need to be able to access information in the STUDENT table, modify this information and add new records to the table. However, they should not delete a record from the database.

Solution: First, you should create user accounts for each operator and then add them to a new role - DataEntry. Next, you will grant them the appropriate permissions, as given below:

```
GRANT SELECT, INSERT, UPDATE  
ON STUDENT  
TO DataEntry
```

And that is all that you need to do. The following example assigns data definition permissions to a role.

Example 2: You want to allow members of the DBA role to add new tables to your database. Furthermore, you want DBA to be able to grant permission to other users.

Solution: The SQL statement to do so is:

```
GRANT CREATE TABLE  
TO DBA  
WITH GRANT OPTION
```

Notice that we have included the WITH GRANT OPTION line to ensure that our DBAs can assign this permission to other users.

Let us now look at the commands for removing permissions from users.

Removing Permissions

Once we have granted permissions, it may be necessary to revoke them at a later date. SQL provides us with the REVOKE command to remove granted permissions. The following is the syntax of this command:

```
REVOKE [GRANT OPTION FOR] <permissions>  
ON <table>  
FROM <user/role>
```

Please notice that the syntax of this command is almost similar to that of the GRANT command. Please also note that the WITH GRANT OPTION is specified on the REVOKE command line and not at the end of the command as was the case in GRANT. As an example, let us imagine we want to revoke a previously granted permission to the user Usha, such that she is not able to remove records from the STUDENT database. The following SQL command will be able to do so:

```
REVOKE DELETE  
ON STUDENT  
FROM Usha
```

The access control mechanisms supported by the SQL is a good starting point, but you must look into the DBMS documentation to locate the enhanced security measures supported by your system. You will find that many DBMS support more advanced access control mechanisms, such as granting permissions on specific attributes.

SQL does not have very specific commands for recovery but, it allows explicit COMMIT, ROLLBACK and other related commands.

11.6 AUDIT TRAILS IN DATABASES

One of the key issues to consider while procuring a database security solution is making sure you have a secure audit trail. An audit trail tracks and reports activities around confidential data. Many companies have not realised the potential amount of risk associated with sensitive information within databases unless they run an internal audit which details who has access to sensitive data and have assessed it. Consider the situation in which a DBA who has complete control of database information may conduct a security breach with respect to business details and financial information. This will cause tremendous loss to the company. In such a situation database audit helps in locating the source of the problem. The database audit process involves a review of log files to find and examine all reads and writes to database items during a specific time period to ascertain mischief, if any. A banking database is one such database which contains very critical data and should have the security feature of auditing. An audit trail is a log that is used for the purpose of security auditing.

Database auditing is one of the essential requirements for security, especially for companies in possession of critical data. Such companies should define their auditing strategy based on their knowledge of the application or database activity. Auditing need not be of the type “all or nothing”. One must do intelligent auditing to save time and reduce performance concerns. This also limits the volume of logs and also causes more critical security events to be highlighted.

More often than not, it is the insiders who make database intrusions as they often have network authorisation, knowledge of database access codes and the idea about the value of data they want to exploit. Sometimes, despite having all the access rights and policies in place, database files may be directly accessible (either on the server or from backup media) to such users. Most database applications store information in ‘form text’ that is completely unprotected and viewable.

As huge amounts are at stake, incidents of security breaches will increase and continue to be widespread. For example, a large global investment bank conducted an audit of its proprietary banking data. It was revealed that more than ten DBAs had unrestricted access to their key sensitive databases, and over a hundred employees had administrative access to the operating systems. The security policy that was in place was that proprietary information in the database should be denied to employees who did not require access to such information to perform their duties. Further, the bank’s database internal audit also reported that the backup data (which is taken once every day) also caused concern as backup media could get stolen. Thus, the risk to the database was high and real and the bank needed to protect its data.

However, a word of caution, while considering ways to protect sensitive database information, please ensure that the privacy protection process should not prevent authorised personnel from obtaining the right data at the right time.

Credit card information is the single most common financially traded information that is desired by database attackers. The positive news is that database misuse or unauthorised access can be prevented with currently available database security products and audit procedures.



Check Your Progress 3

- 1) On what systems does the security of a Database Management System depend?

.....

- 2) Write the syntax for granting permission to alter the database.

.....

- 3) Write the syntax for ‘Revoke Statement’ that revokes the grant option.

.....

- 4) What is the main difference between data security and data integrity?

.....

.....

11.7 SUMMARY

In this unit, we have discussed the recovery of the data contained in a database system after failure. Database recovery techniques are methods of making the database fault tolerant. The aim of the recovery scheme is to allow database operations to be resumed after a failure with no loss of information and at an economically justifiable cost. The basic technique to implement database recovery is to use data redundancy in the form of logs and archival copies of the database. Checkpoint helps the process of recovery.

Security and integrity concepts are crucial. The DBMS security mechanism restricts users to only those pieces of data that are required for the functions they perform. Security mechanisms restrict the type of actions that these users can perform on the data that is accessible to them. The data must be protected from accidental or intentional (malicious) corruption or destruction.

Security constraints guard against accidental or malicious tampering with data; integrity constraints ensure that any properly authorised access, alteration, deletion, or insertion of the data in the database does not change the consistency and validity of the data. Database integrity involves the correctness of data, and this correctness has to be preserved in the presence of concurrent operations. The unit also discussed the use of audit trails.

11.8 SOLUTIONS/ANSWERS

Check Your Progress 1

- 1) Recovery is needed to take care of the failures that may be due to software, hardware and external causes. The aim of the recovery scheme is to allow database operations to be resumed after a failure with the minimum loss of information and at an economically justifiable cost. One of the common techniques is log-based recovery. The transaction is the basic unit of recovery.
- 2) All recovery processes require redundancy. Log-based recovery process records the consistent state of the database and all the modifications made by a transaction into a log on the stable storage. In case of any failure, the stable log and the database states are used to create a consistent database state.
- 3) A checkpoint is a point when all the database updates and logs are written to stable storage. A checkpoint ensures that not all the transactions need to be REDONE or UNDONE. Thus, it helps in faster recovery from failure. The checkpoint helps in recovery, as in case of a failure all the committed transactions prior to the checkpoint are NOT to be redone. Only non-committed transactions at the time of checkpoint or transactions that started after the checkpoint are required to be REDONE or UNDONE based on the log.

Check Your Progress 2

- 1) The deferred database modification recovery algorithm postpones, as far as possible, the writing of updates into the physical database. Rather, the modifications are recorded in the log file, which is written into stable storage. In case of a failure, the log can be used to REDO the committed transactions. In this process UNDO operation is not performed.
- 2) The log of deferred database modification technique just stores the REDO information. UNDO information is not required as updates are performed in the main memory buffers only, while the immediate modification scheme maintains both the UNDO and REDO information in the log.
- 3) Buffer management is important from the point of view of recovery, as it may determine the time taken in recovery. It is also used for implementing different recovery algorithms.
Remote backup is very useful in the case of disaster recovery. It may also make the database available even if one site of the database fails.

Check Your Progress 3

- 1) The database system security may depend on the security of the Operating system, including the network security; and the application that uses the database and services that interact with the web server.
- 2) GRANT ALTER DATABASE TO UserName
- 3) REVOKE GRANT OPTION
FOR <permissions> ON <table> FROM <user/role>
- 4) Data security is the protection of information that is maintained in the database against unauthorised access, modification or destruction. Data integrity is the mechanism that is applied to ensure that data in the database is correct and consistent.

UNIT 12 QUERY PROCESSING AND EVALUATION

Structure	Page Nos.
12.0 Introduction	
12.1 Objectives	
12.2 Query Processing: An Introduction	
12.2.1 Role of Relational Algebra in Query Optimisation	
12.2.2 Using Statistics and Stored Size for Cost Estimation.	
12.3 Cost of Selection Operation	
12.3.1 File scan	
12.3.2 Index scan	
12.3.3 Implementation of Complex Selections	
12.4 Cost of Sorting	
12.5 Cost of Join Operation	
12.5.1 Block Nested-Loop Join	
12.5.2 Merge-Join	
12.5.3 Hash-Join	
12.6 Other Operations	
12.7 Representation and Evaluation of Query Expressions	
12.7.1 Evaluating a Query Tree.	
12.7.2 Evaluating Complex Joins	
12.8 Creation of Query Evaluation Plans	
12.8.1 Transformation of Relational Expressions	
12.8.2 Query Evaluation Plans	
12.8.3 Choosing an Optimal Evaluation Plan	
12.8.4 Cost and Storage-Based Query Optimisation	
12.9 View and Query Processing	
12.9.1 Materialised View	
12.9.2 Materialised Views and Query Optimisation	
12.10 Summary	
12.11 Solutions/Answers	

12.0 INTRODUCTION

The Query Language – SQL is one of the main reasons for the success of RDBMS. A user just needs to write the query in SQL that is close to the English language and does not need to say how such a query is to be evaluated. However, a query needs to be evaluated efficiently by the DBMS. But how is a query evaluated efficiently? This unit attempts to answer this question. The unit covers the basic principles of query evaluation, the cost of query evaluation, the evaluation of join queries, etc. in detail. It also provides information about query evaluation plans and the role of storage in query evaluation and optimisation. This unit introduces you to the complexity of query evaluation in DBMS.

12.1 OBJECTIVES

After going through this unit, you should be able to:

- Explain the measure of query cost;
- define algorithms for individual relational algebra operations;
- create and modify query expression;
- define evaluation plan choices, and
- define query processing using views.

12.2 QUERY PROCESSING: AN INTRODUCTION

Before defining the measures of query cost, let us begin by defining query processing. *Figure 1* shows the steps of query processing.

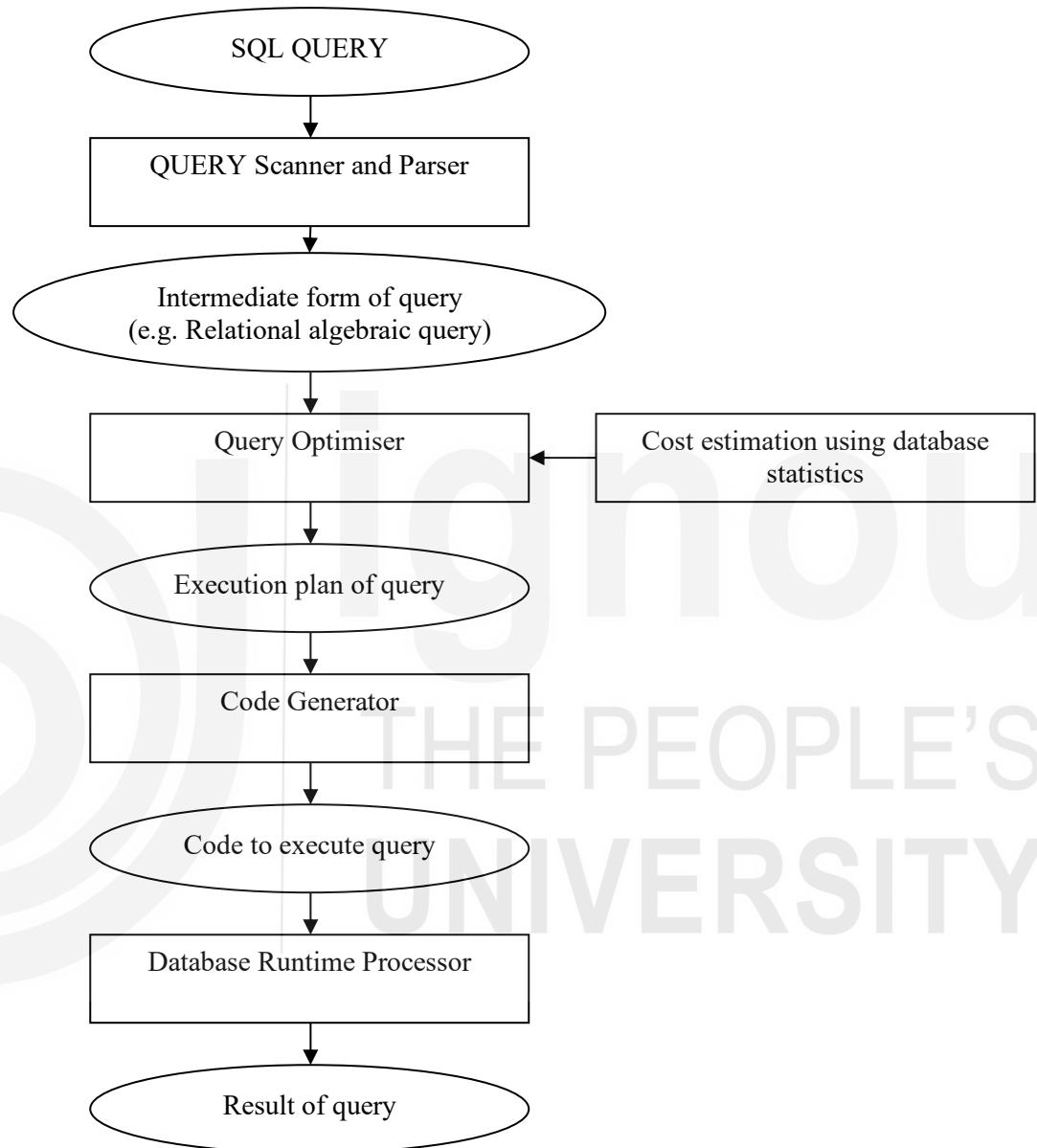


Figure 1: Query processing

In the first step, a query is scanned, parsed, validated and translated into an internal form. This internal form may be the relational algebra (an intermediate query form). The parser checks the syntax of the query and verifies the relations. Next, the query is optimised, and a query execution plan is generated, which is then compiled into a code that can be executed by the database runtime processor. The query processing involves the study of the following concepts:

- measures to find query cost.
- algorithms for evaluating relational algebraic operations.

- evaluating a complete expression using algorithms on individual operations.

12.2.1 Role of Relational Algebra in Query Optimisation

In order to optimise the evaluation of a query, first, you must define the query using relational algebra. A relational algebra expression may have many equivalent expressions. For example, the relational algebraic expression $\sigma_{(\text{salary} < 5000)}(\pi_{\text{salary}}(\text{EMP}))$ is equivalent to $\pi_{\text{salary}}(\sigma_{\text{salary} < 5000}(\text{EMP}))$. This may result in generating many alternative ways of evaluating the query.

Further, a relational algebraic expression can be evaluated in many different ways. A detailed evaluation strategy for an expression is known as an evaluation plan. For example, you can use an index on *salary* to find employees with $\text{salary} < 5000$, or you can perform a complete relation scan and discard employees with $\text{salary} \geq 5000$. Both of these are separate evaluation plans. The basis of the selection of the best evaluation plan is the cost of these evaluation plans.

Query Optimisation: The query optimisation selects the query evaluation plan with the lowest cost among the equivalent query evaluation plans. Cost is estimated using the statistical information obtained from the database catalogue, viz., the number of tuples in each relation, the size of tuples, different values of an attribute, etc. The cost estimation is made on the basis of heuristic rules.

What is the basis for measuring the query cost? The next section addresses this question.

12.2.2 Using Statistics and Stored Size for Cost Estimation

The query cost is generally measured as the total elapsed time for answering the query. There are many factors that contribute to the cost in terms of elapsed time. These are the time of *disk accesses*, *CPU time*, and *data communication time on the network*. However, these times can be measured when the query is being executed. Therefore, you may use statistics, like the number of records, number of blocks, number of attributes, possible number of different values for each attribute, etc., to estimate the cost. However, disk access is typically the predominant cost as disk transfer is very slow. In addition, disk accesses are relatively easier to estimate. Therefore, the following disk access cost can be used to estimate the query cost:

- Number of seeks \times average-seek-time; and
- Number of blocks read \times average-block-read-time; and
- Number of blocks written \times average-block-write-time.

Please note that the cost of writing a block is higher than the cost of reading a block. This is due to the fact that the data is read back after being written to ensure that the write operation was successful. However, for the sake of simplicity, we will just use the *number of block transfers from the disk as the cost measure*. We will also ignore the difference in cost between sequential and random Input/Output, which **depends** on the search criteria, such as point/range query on an ordering field vs other fields; and the file structures: heap, sorted, hashed. In addition, the number of block transfer is also dependent on the use of indices, such as primary, clustering, secondary, B+ tree, etc. Other cost factors may include buffering, disk placement, view materialisation, overflow / free space management, etc. Please also note that CPU time and communication time are also not being considered for cost computation.

In the subsequent section, let us try to find the cost estimates of various operations. Please note that the stored size of a relation is an important cost estimate, if the entire database-related cost is to be estimated. The other statistics like the number of tuples or the number of attributes etc. are useful when only part of the database is to be considered for query processing.

12.3 COST OF SELECTION OPERATION

The selection operation can be performed in several ways. Let us discuss the algorithms and the related cost of performing selection operation.

12.3.1 File scan

File scan algorithms locate and retrieve records that fulfil a selection condition in a file. The following are the two basic file scan algorithms for selection operation:

- 1) *Linear search:* This algorithm scans each file block and tests all records to see whether their attributes match the selection condition.

The cost of this algorithm (in terms of block transfer): This algorithm would require reading all the blocks of the file, as it must test all the records for the specific condition.

Cost *To find records that match a given criteria*

$$\begin{aligned} &= \text{Size of database in terms of Number of blocks} \\ &= N_b. \end{aligned}$$

Cost *Finding a specific value of key attribute*

$$\begin{aligned} &= \text{Average number of block transfer for locating the value} \\ &\quad (\text{on an average half of the file needs to be traversed}) \text{ so the cost is} \\ &= N_b/2. \end{aligned}$$

Linear search can be applied regardless of selection condition or ordering of records in the file, or availability of indices.

- 2) *Binary search:* It is applicable when the selection is an equality comparison on an attribute on which the file is ordered. Assume that the blocks of a relation are stored continuously then, the cost can be estimated as:

Cost *binary search*

$$\begin{aligned} &= \text{Cost}_{\text{Block based Binary search to find first tuple that matches the search criteria}} \\ &+ \text{Cost}_{\text{fetching contiguous blocks till the records keep matching the criteria}} \end{aligned}$$

$$= [\log_2 (N_b)] + \frac{\text{Mean of the number of tuples with similar attribute values}}{\text{Size of a Block of data in terms of Number of tuples}}$$

You may observe that the cost computation is based on the database statistics.

12.3.2 Index scan

The index scan can be used for cases where the database contains an index on an attribute set that forms the search key.

- 1) (a) *Scanning for equality condition on a Primary index:* These kinds of searches try to find a specific key value using the primary index of a database system. Since the search criteria include equality on the primary key, therefore, the output of this search would be just a single record or no record at all. The cost of the scan is defined as:

Cost = The depth traversed in the index to locate the block pointer + 1 (for transfer of block consisting of desired primary key value).

- (b) *Hash key:* It retrieves a single block directly, thus, the cost in the hash key organisation is given as:

=Block transfer needed for finding hash target +1

- 2) *Primary index-scan for comparison:* Assuming that the relation is sorted on the attribute(s) that are being compared, ($<$, $>$ etc.), then we need to locate the first record satisfying the condition after which the records are scanned forward or backwards as the condition may be, displaying all the records. Thus, the cost, in this case, would be:

$$\text{Cost} = \text{Number of block transfers to locate the value in index} + \\ \text{Transferring all the blocks of data satisfying that condition.}$$

Please note you can roughly compute the number of blocks satisfying the condition as:

$$\begin{aligned} &\text{Number of attribute values that satisfy the condition} \\ &\quad \times \text{average number of tuples per attribute value} \\ &\quad / \text{blocking factor of the relation.} \end{aligned}$$

- 3) (a) *Equality on search key of secondary index:* Retrieves a single record if the search key is a candidate key.

$$\text{Cost} = \text{cost of accessing index} + 1.$$

It retrieves multiple records if the search key is not a candidate key.

$$\begin{aligned} \text{Cost} &= \text{cost of accessing index} + \text{number of records retrieved} \\ &\quad (\text{It can be very expensive}). \end{aligned}$$

Each record may be on a different block, thus requiring one block access for each retrieved record. This is the worst-case cost.

(b) *Secondary index comparison:* For the queries of the type that use the comparison on *secondary index value* $>$ *value searched*, the index is used to find the first index entry which is greater than the *value searched*, thereafter, the indexed is scanned sequentially till the end, finding the pointers to records.

For the \leq type query, just scan the leaf pages of the index to find the pointers to the records until the first entry that satisfies the condition is found. Thereafter, the index can be scanned till the records satisfy the condition to obtain pointers to the records.

You may please note that after you have found the pointers to the records using the index scan, retrieving those pointed records may require one block transfer for each record. Please note that linear file scans may be cheaper if many records are to be fetched.

12.3.3 Implementation of Complex Selections

Conjunction: Conjunction is basically a set of AND conditions.

Conjunctive selection using one index: In such case, select any algorithm given earlier on one or more conditions and then test remaining conditions on the selected tuples after fetching them into the memory buffer.

Conjunctive selection using the multiple-key index: Use appropriate composite (multiple-key) index if they are available.

Disjunction: Disjunctions are basically a set of OR conditions.

Disjunction using the union of identifiers is applicable if *all* conditions have available indices, otherwise, use linear scan. Use the corresponding index for

each condition, take the union of all the obtained sets of record pointers, and eliminate duplicates, then fetch data from the file.

Negation: Use linear scan on file. However, if very few records are available in the result and an index is applicable on an attribute, which is being negated, then find the satisfying records using the index and fetch them from the file.

12.4 COST OF SORTING

This section introduces the cost of query evaluation when it requires sorting of records. There are various methods that can be used in the following ways:

- 1) Use an existing applicable ordered index (e.g., B+ tree) to read the relation in sorted order.
- 2) Build an index on the relation, and then use the index to read the relation in sorted order. (Options 1 and 2 may lead to one block access per tuple).
- 3) Techniques like *quicksort* can be used for relations that fit in the memory.
- 4) *External sort-merge* is a good choice for relations that do not fit in the memory.

Once you decide on the sorting technique, you can find the cost of these algorithms to find the sorted file. You may refer to further readings for more details.

☞ Check Your Progress 1

- 1) What are the basic steps in query processing?

.....

.....

- 2) How can the cost of a query be measured?

.....

.....

- 3) What are the various methods adopted for performing selection operation?

.....

.....

12.5 COST OF JOIN OPERATION

There are several algorithms that can be used to implement joins:

- Nested-loop join
- Block nested-loop join
- Indexed nested-loop join
- Merge-join
- Hash-join

The choice of join algorithm is based on the cost estimates. We will elaborate on only a few of these algorithms in this section. The following relations and related statistics will be used to elaborate those algorithms.

MARKS (enrollno, subjectcode, marks): 20000 rows, 500 blocks

STUDENT (enrollno, name, dob): 5000 rows, 200 blocks.

12.5.1 Block Nested-Loop Join

In this join approach, a complete block of the outer loop is joined with the complete block of the inner loop. We have chosen STUDENT as the outer relation, as it is smaller in size and, therefore, will result in a smaller number of overall block transfers.

The algorithm for this may be written as:

```
for each tuple  $s_i$  in block  $s$  of STUDENT
{
    for each tuple  $m_i$  in block  $m$  of MARKS
    {
        Check if  $s_i$  and  $m_i$  satisfy the join condition
        if they do output joined tuple to the result
    };
};
```

In the worst case, when only one block of both the relations can be stored in RAM, the estimation of block accesses is:

$$\begin{aligned} &= \text{Number of Blocks of outer relation (STUDENT)} \times \text{Number of blocks of inner relation (MARKS)} + \text{Number of blocks of outer relation (STUDENT)}. \\ &= 200 \times 500 + 200 = 100200 \end{aligned}$$

In the best case, when the smaller relation can be completely in RAM, the number of block accesses would be = Blocks of STUDENT + Blocks of MARKS

$$= 200 + 500 = 700$$

Improvements to Block Nested-Loop Algorithm

The following modifications improve the block Nested method:

- Use $M - 2$ disk blocks as the blocking unit for the outer relation, where $M =$ memory size in terms of the number of blocks.
- Use one buffer block to buffer the inner relation.
- Use one buffer block to buffer the output.

This method minimises the number of iterations.

12.5.2 Merge-Join

The merge-join is applicable to equijoin and natural join operations only. It has the following process:

- 1) Sort both relations on the joining attribute (if not already sorted).
- 2) Merge the sorted relations to Join them. In this step, every pair with the same value on the joining attribute must be matched.

For example, consider the instances of STUDENT and MARKS relations, as given in Figure 2.

STUDENT		
enrolno	Name
1001	Ajay
1002	Aman
1005	Rakesh
1100	Raman

Block 1 of STUDENT relation

MARKS		
enrolno	subjectcode	Marks
1001	MCS-211	55
1001	MCS-212	75
1002	MCS-212	90
1005	MCS-215	75

Block 1 of MARKS relation

Figure 2: Sample Relations for Computing Join

Computation of the number of block accesses:

Join operation on enrolment number would be as follows:

- i) The enrolment number 1001 of STUDENT relation will join with two tuples of MARKS relation as:

1001 1001	MCS-211
1001 1001	MCS-212

- ii) Joining on the key 1001 will be over as soon as you find 1002 in MARKS relations, as MARKS relation is also sorted on enrolno. Thus, the join will continue as Merge-Join operations and the following tuples will be output after the output of the first two tuples for 1001.

This will be followed by output.

1002 1002	MCS-212
1005 1005	MCS-215

You may observe that the cost of the Merge-Join operation can be computed based on the assumption that a block that is part of the merge is read only once. The basic underlying assumption is that the main memory is sufficiently large to accommodate all the tuples of a specific attribute join value. Therefore, the number of block accesses for Merge-Join is:

$$= \text{Blocks of STUDENT} + \text{Blocks of MARKS} + \text{the cost of sorting on enrolno} \\ (\text{if relations are unsorted})$$

12.5.3 Hash-Join

Hash-join can be performed for both the equijoin and natural join operations. A hash function h is applied on joining attributes to partition tuples of both relations. In the case of STUDENT and MARKS relations, the hash function h maps joining attribute (enrolno in our example case) values to $\{0, 1, \dots, n-1\}$.

The join attribute is hashed to the join-hash partitions. In the example of *Figure 4*, we have used the mod 5 function for hashing, therefore, $n = 05$.

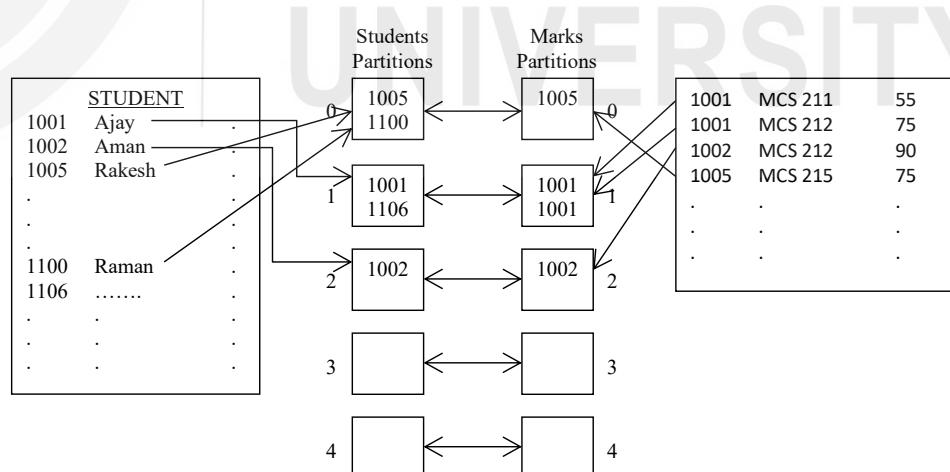


Figure 3: A hash-join example

Once the partition tables of STUDENT and MARKS are made on the enrolno, then only the corresponding partitions will participate in the join as:

A STUDENT tuple and a MARKS tuple that satisfy the Join condition will have the same value for the join attributes. Therefore, they will be hashed to an equivalent partition and, thus, can be joined easily. For example, STUDENT partition 1 consists of tuples of STUDENT with enrolno 1001, 1006; whereas

MARKS partition 1 consists of marks of the student enrolno 1001 in two different subjects. The join can now simply be performed for enrolno 1001. There is no joining MARKS tuple for STUDENT 1006.

Algorithm for Hash-Join

The hash-join of two relations r and s is computed as follows:

- Partition the relation r and s using the same hash function h . (When partitioning a relation, one block of memory is reserved as the output buffer for each partition). It may be noted that the tuples in i^{th} partition of r may join only with the tuples of i^{th} partition of s .
- For each partition s_i of s , load the partition into memory and build an in-memory hash index using the joining attribute. Why is this hash index needed? As several joining attributes may get mapped into the same partition. For example, partition 0 and partition 1 in Figure 3.
- Read the partition i of r into memory block by block. For each tuple in r_i , find the tuples s_i in the hash index of s_i , which would join with that tuple in r_i . Output the joined results.

The value n (the number of partitions) and the hash function h are chosen in such a manner that each s_i should fit into the memory. Typically, n is chosen as:

$$[\text{the Number of blocks of } s / \text{Number of memory buffers}] \times f$$

where f is typically around 1.2.

The partition of r can be large, as r_i need not fit in memory.

You may refer to the further readings for more details on hash join.

Cost calculation for Simple Hash-Join

- Cost of partitioning r and s :* all the blocks of r and s are read once and, after partitioning, written back to partitions, so

$$\text{cost1} = 2(\text{blocks of } r + \text{blocks of } s).$$
- The cost of performing the hash-join using the hash index on s will require at least one block transfer for reading the partitions.

$$\text{cost2} = (\text{blocks of } r + \text{blocks of } s)$$
- There are a few more blocks in the main memory that may be used for evaluation, they may be read or written back. We ignore this cost as it will be too low in comparison to cost1 and cost2 .
Thus, the total $\text{cost} = \text{cost1} + \text{cost2}$

$$= 3(\text{blocks of } r + \text{blocks of } s)$$

Even if s is recursively partitioned, *hash-table overflow* can occur, i.e., some partition s_i may not fit in the memory. This may happen if many tuples in s have the same value for join attributes or the chosen hash function is bad. Partitioning is said to be *skewed* if some partitions have significantly more tuples than others. This is the overflow condition. The overflow can be handled in a variety of ways; however, they are beyond the scope of this unit.

Let us explain the hash join and its cost for the natural join $\text{STUDENT} \bowtie \text{MARKS}$. Assume a memory size of 25 blocks $\Rightarrow M=25$.

SELECT s as STUDENT as it has a smaller number of blocks (200 blocks) and r as MARKS (500 blocks).

$$\begin{aligned}\text{Number of partitions to be created for STUDENT} &= (\text{blocks of STUDENT}/M) \times 1.2 \\ &= (200/25) \times 1.2 = 9.6 \approx 10\end{aligned}$$

Thus, the STUDENT relation will be partitioned into 10 partitions of 20 blocks each. MARKS will also have 10 partitions of 50 blocks each. The 25 buffers will be used as:
 20 blocks for one complete partition of STUDENT,
 01 block will be used for input of MARKS partitions, and
 The remaining 4 blocks may be used for storing the results.

The total cost = $3(200+500) = 2100$ as no recursive partitioning is needed.

12.6 OTHER OPERATIONS

There are many other operations that are performed in database systems. Let us introduce these operations in this section.

Duplicate Elimination: Duplicate elimination may be implemented by using hashing or sorting. On sorting, duplicates will be adjacent to each other thus, may be identified and deleted. An optimised method for duplicate elimination can be the deletion of duplicates during generation as well as at intermediate merge steps in an external sort-merge. Hashing is similar – duplicates will be clubbed together in the same bucket and, therefore, may be eliminated easily.

Projection: It may be implemented by performing the projection on each tuple, followed by duplicate elimination.

Aggregate Function Execution: Aggregate functions can be implemented in a manner similar to duplicate elimination. Sorting or hashing can be used to bring tuples in the same group together, and then aggregate functions can be applied to each group. For count, min, max, and sum, you may add up the aggregates. For calculating the average, take the sum of the aggregates and count the number of aggregates; and then divide the sum with the count at the end.

Set operations (such as \cup and \cap) can either use a variant of merge-join after sorting or a variant of hash-join.

Using the Hashing:

- 1) Partition both relations using the same hash function, thereby creating partitions consisting of tuples of r and s , as:

$$r_0, r_1 \dots r_{n-1} \text{ and } s_0, s_1 \dots s_{n-1}$$

- 2) Process each partition i as follows:

Using a different hashing function, build an in-memory hash index on r_i after it is brought into the memory.

$r \cup s$: Add tuples in s_i to the hash index if they are not already in it.
 Output the hash index.

$r \cap s$: For each tuple s_i of the relation s , check if it is in the hash index, if yes, output this tuple.

$r - s$: For each tuple in s_i of the relation s , if the similar r_j is part of the hash index, delete r_j from the hash index. Once the entire s is processed, output the remaining hash index tuples.

There are many other operations as well. You may wish to refer to them in further readings.



Check Your Progress 2

- 1) Define the algorithm for Block Nested-Loop Join for the worst-case scenario.

.....
.....

- 2) What is the cost of Hash-Join?

.....
.....

- 3) What are the other operations that may be part of query evaluation?

.....
.....

12.7 REPRESENTATION AND EVALUATION OF QUERY EXPRESSIONS

Before we discuss the evaluation of a query expression, let us briefly explain how a SQL query may be represented. Consider the following STUDENT and MARKS relations:

STUDENT (enrolno, name, phone)

MARKS (enrolno, subjectcode, grade)

To find the result of the student(s) whose phone number is '1129250025', the following SQL query may be written:

```
SELECT enrolno, name, subjectcode, grade  
FROM STUDENT s, MARKS m  
WHERE s.enrolno=m.enrolno AND phone='1129250025'
```

The equivalent relational algebraic query for this would be:

$$\pi_{\text{enrolno, name, subjectcode, grade}} ((\sigma_{\text{phone}='1129250025'} (\text{STUDENT}) \bowtie \text{MARKS}))$$

This is a very good internal representation; however, it may be a good idea to represent the relational algebraic expression as a query tree on which algorithms for query optimisation can be designed easily. In a query tree, nodes are the operators, and relations represent the leaf. The query tree for the relational expression above would be:

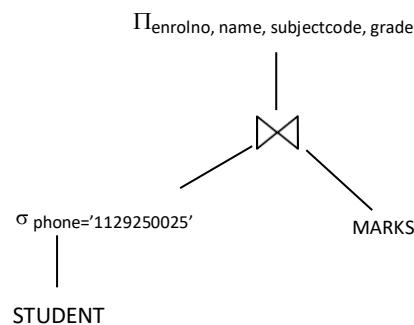


Figure 4: A Sample query tree

12.7.1 Evaluating a Query Tree.

In this section, let us examine the methods for evaluating a query expression that is expressed as a query tree. In general, we use two methods:

- Materialisation
- Pipelining.

Materialisation: Evaluates a relational algebraic expression in a bottom-up approach by explicitly generating and storing the results of each operation of expression. For example, for *Figure 4*, first selection operation on the STUDENT relation would be performed. Next, the result of the selection operation would be joined with the MARKS relation. Finally, the projection operation will be performed. Materialised evaluation is always possible even though the cost of writing/reading results to/from disk can be quite high.

Pipelining: Evaluates operations in a multi-threaded manner (i.e., passes tuples output from one operation to the next parent operation as input) even as the first operation is being executed. In the previous expression tree, it does not store (or materialise) results instead, it passes the tuples of the selection operation directly to the join operation. Similarly, it does not store the results of the join operation and passes the tuples of join operations directly to the projection operation. Thus, there is no need to store temporary relations on a disk for each operation. Pipelining may not always be possible or easy if sort or hash-join operations are used.

The pipelining execution method may involve a buffer, which is being filled by the result tuples of a lower-level operation, while records may be picked up from the buffer by a higher-level operation.

12.7.2 Evaluating Complex Joins

When an expression involves three relations, then you have more than one strategy for the evaluation of the expression. For example, join of relations such as STUDENT \bowtie MARKS \bowtie SUBJECTS may involve the following three strategies:

Strategy 1: Compute STUDENT \bowtie MARKS, and *join* the result with SUBJECTS.

Strategy 2: Compute MARKS \bowtie SUBJECTS first, and then *join* the result with STUDENT.

Strategy 3: Perform the pair of joins at the same time. This can be done by building an index of enrolno in STUDENT and on subjectcode in SUBJECTS. For each tuple m in MARKS, look up the corresponding tuples in STUDENT and the corresponding tuples in SUBJECTS. Each tuple of MARKS will be examined only once. Strategy 3 combines two operations into one special-purpose operation that may be more efficient than implementing the joins of two relations.

12.8 CREATION OF QUERY EVALUATION PLANS

We have already discussed query representation and its evaluation in the earlier section of this unit, but can something be done during these two stages that optimise the query evaluation? This section deals with this process in detail.

Generation of query-evaluation plans for a query expression involves several steps:

- 1) Generating logically equivalent expressions using **equivalence rules**
- 2) Generate alternative query plans.
- 3) Choose the cheapest plan based on the estimated cost.

The overall process is called **cost-based optimisation**. The cost difference between a good and a bad method of evaluating a query would be enormous. We need to estimate the cost of operations and statistical information about relations. For example, the number of tuples, the number of distinct values for an attribute, etc., helps estimate the size of the intermediate results, and information like available indices may help in estimating the cost of complex expressions. Let us discuss all the steps in query-evaluation plan development in more detail.

12.8.1 Transformation of Relational Expressions

Two relational algebraic expressions are said to be **equivalent** if, on every legal database instance, the two expressions generate the same set of tuples (the order of tuples is irrelevant). “Appendix-A”, given at the end of this unit, lists the rules that may be used to generate equivalent relational expressions. However, the rules given in Appendix A are too general, and a few heuristics rules generated from these rules can be used to transform the relational expressions. These rules are:

- (1) Combine a cascade of selections into a conjunction and test all the predicates on the tuples in a single iteration:
 Expression like: $\sigma_{02}(\sigma_{01}(E))$
 Can be converted to: $\sigma_{02 \wedge 01}(E)$
- (2) Combining a cascade of projections into a single outer projection (please note that the final projection would be the outer projection).
 Expression like: $\pi_4(\pi_3(\dots(E)))$
 Can be converted to: $\pi_4(E)$
- (3) Commute the selection and projection or vice-versa. This commutation may sometimes reduce query cost.
- (4) Use associative or commutative rules for the Cartesian product or join operation to find various alternative paths for query evaluation.
- (5) Move the selection and projection (projection may be expanded to include join condition) before Join operations. The selection and projection result in the reduction of the number of tuples and, therefore, may reduce the cost of joining.
- (6) Commute the projection and selection with Cartesian product or union.

Let us explain the use of some of these rules with the help of an example. Consider the query for the relations:

STUDENT (enrolno, name, phone)
 MARKS (enrolno, subjectcode, grade)
 SUBJECT (subjectcode, sname)

Example 1: Consider the query: Find the enrolment number, name, and grade of those students who have secured an A grade in the subject DBMS. One of the possible solutions to this query may be:

$\pi_{\text{enrolno, name, grade}}(\sigma_{(\text{sname} = 'DBMS' \wedge \text{grade} = 'A')}((\text{STUDENT} \bowtie \text{MARKS}) \bowtie \text{SUBJECT}))$
 The query tree for this would be:

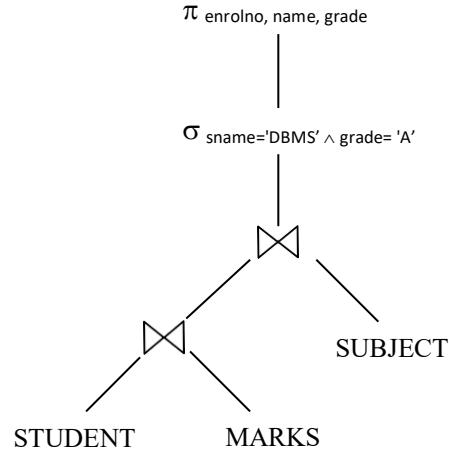


Figure 5: The query tree of example 1

As per the suggested rules, the selection condition may be moved before the join operation. The selection condition given in *Figure 5* above is: $sname = 'DBMS'$ and $grade = 'A'$. Both of these conditions belong to different tables, as $sname$ is available only in the SUBJECT table and $grade$ in the MARKS table. Thus, the selection conditions will be mapped accordingly, as shown in *Figure 6*. Thus, the equivalent expression will be:

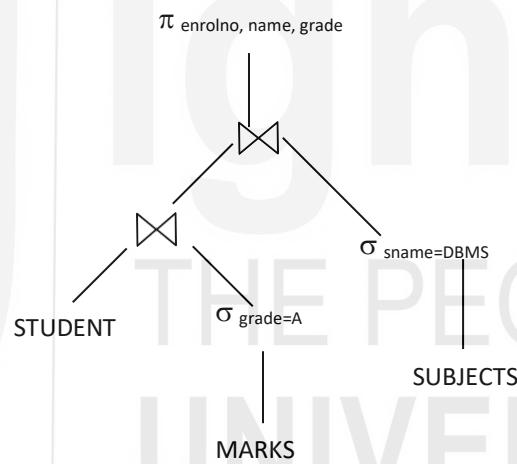


Figure 6: A modified tree

Further, the expected size of SUBJECT and MARKS after selection will be small, so it may be a good idea to join MARKS with SUBJECT first, and thereafter, the resultant relation is joined with the STUDENT relation. Hence, the associative law of JOIN may be applied.

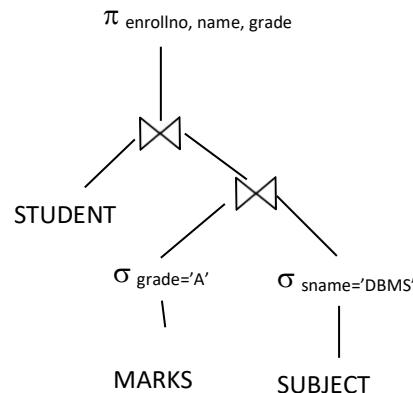


Figure 7: Modified query tree using associativity of join.

Even moving the projection before possible join, wherever possible, may optimise the query processing, as projection may also reduce the size of the intermediate result. Thus, you can move projection of outer join (refer to Figure 7) to inner join (Refer Figure 8).

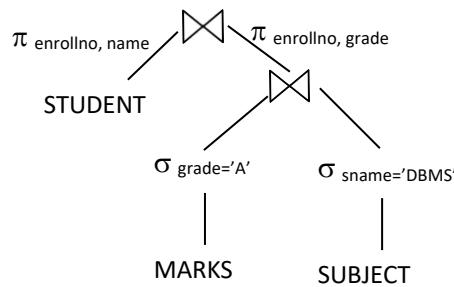


Figure 8: Moving the projection.

The final equivalent relation algebraic query expression of query of example 1 is:

$$(\pi_{\text{enrolno}, \text{name}}(\text{STUDENT})) \bowtie (\pi_{\text{enrolno}, \text{grade}}((\sigma_{\text{grade}=\text{'A'}}) \text{MARKS}) \bowtie (\sigma_{\text{sname}=\text{'DBMS'}}) \text{SUBJECT}))$$

Alternative Query Expressions

A relational algebraic query can be optimised by the query optimiser, which creates several relational algebraic expressions that are equivalent to the query expression using the equivalence rules of relational algebra. One of the techniques to do so is to keep applying the equivalence rules to the relational algebraic query to generate a new set of expressions. However, this is a very time-consuming process. Therefore, in general, a set of heuristics, which are based on certain criteria like “apply those transformation rules that result in a reduction in the size of intermediate results”, can be used with the objective to produce more efficient equivalent query expressions.

12.8.2 Query Evaluation Plans

Let us first define the term Evaluation Plan. An evaluation plan defines exactly which algorithm is to be used for each operation and how the execution of the operation is coordinated. For example, *Figure 9* shows the query tree with an evaluation plan.

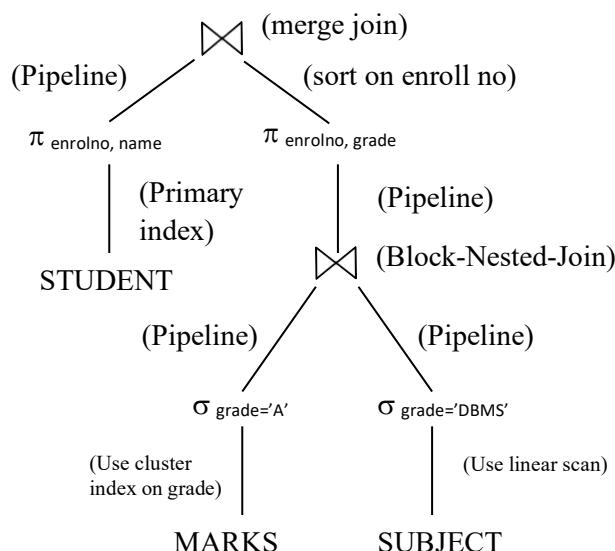


Figure 9: A sample query evaluation plan

12.8.3 Choosing an Optimal Evaluation Plan

The output of the previous step of query evaluation is the number of query evaluation plans. Which of these plans should be chosen for the final query evaluation? This decision is normally based on the database statistics. In addition, it is not necessary that individual best plans for each part of query evaluation may make the best query evaluation algorithm. For example, to join two relations, hash join is computationally less expensive than the merge join; however, the merge join produces sorted output. Thus, for the cases where sorted output may be useful for further processing of part results, merge join may be preferred. Similarly, the use of the nested loop method of joining may allow pipelining of the results of the join operation for further operations. In general, for choosing an optimal evaluation plan, you may perform cost-based optimisation to choose an optimal query evaluation plan.

12.8.4 Cost and Storage-Based Query Optimisation

Cost-based optimisation is performed on the basis of the cost of various individual operations that are to be performed as per the query evaluation plan. The cost is calculated as we have explained in section 12.3 with respect to the method and operation (JOIN, SELECT, etc.).

Storage and Query Optimisation

Cost calculations are primarily based on disk access; thus, storage has an important role to play in cost computation. In addition, some of the operations also require intermediate storage; thus, the cost is further enhanced in such cases. The cost of finding an optimal query plan is offset by the savings in terms of the query-execution time, particularly by reducing the number of slow disk accesses.

12.9 VIEWS AND QUERY PROCESSING

A view is defined as a query. The view may maintain the complete set of tuples following evaluation. This requires a lot of memory space; therefore, it may be a good idea to partially pre-evaluate it.

12.9.1 Materialised View

A materialised view is a view whose contents are computed and stored. Materialising the view would be very useful if the result of a view is required frequently, as it saves the effort of computing the view again.

Further, the task of keeping a materialised view up to date with the underlying data is known as materialised view maintenance. Materialised views can be maintained by recomputation on every update. A better option is to use incremental view maintenance, i.e., where only the affected part of the view is modified. View maintenance, in general, can be performed using triggers, which can be written for any data manipulation operation on the relations that are part of a view definition.

12.9.2 Materialised Views and Query Optimisation

We can perform query optimisation by rewriting queries to use materialised views. For example, assume that a materialised view of the join of two tables b and c is available as:

$$a = b \text{ NATURAL JOIN } c$$

Any query that uses natural join on b and c can use this materialised view ' a ' as:

Consider you are evaluating a query:

$$z = r \text{ NATURAL JOIN } b \text{ NATURAL JOIN } c$$

Then this query would be rewritten using the materialised view ‘a’ as:

$$z = r \text{ NATURAL JOIN } a$$

Do you need to perform materialisation? It depends on cost estimates for the two alternatives viz., use of a materialised view by view definition, or simple evaluation.

Query optimiser should be extended to consider all the alternatives of view evaluation and choose the best overall plan. This decision must be made on the basis of the system workload. Indices in such decision-making may be considered as specialised views. Some database systems provide tools to help the database administrator with index and materialised view selection.

☛ Check Your Progress 3

- 1) List the methods used for the evaluation of expressions.

.....
.....

- 2) How do you define cost-based optimisation?

.....
.....
.....

- 3) Define the term “Evaluation plan”.

.....
.....
.....

12.10 SUMMARY

This Unit introduces you to the basic concepts of query processing and evaluation. A query is to be represented in a standard form before it can be processed. In general, a query is evaluated after representing it using relational algebra. Thereafter, several query evaluation plans are generated using query transformations and an optimal plan is chosen for query evaluation. To find the cost of a query evaluation plan, you may use database statistics. The unit also defines various algorithms and the cost of these algorithms for evaluating various operations like select, project, join etc. You may refer to database textbooks for more details on query evaluation.

12.11 SOLUTIONS/ANSWERS

Check Your Progress 1

- 1) The steps of query evaluation are as follows:
 - a. In the first step, query scanning, parsing and validating is done.
 - b. Next, translate the query into a relational algebraic expression.
 - c. Next, the syntax is checked along with the names of the relations.

- d. Finally, the optimal query evaluation plan is executed and the
 - e. answers to the query are returned.
- 2) Query cost is measured by considering the following activities:
- Number of disk-head seeks.
 - Number of blocks of tables.
 - Number of blocks of the results to be written
- Please note that the cost measures above are based on the *number of disk blocks to memory buffer transfer*. We generally ignore the difference in cost between sequential and random I/O, CPU and communication costs.
- 3) The selection operation can be performed in several ways, such as:
Using File Scan: Using linear search or using the Binary search
Using Index Scan: Using the primary index (for equality) or the Hash key. You can also use a clustering index or a secondary index.

Check Your Progress 2

- 1) For each tuple in block B_r of r {
 - For each tuple in block B_s of s {
 - Test pair(t_i, s_i) to see if they satisfy the join condition
 - If they do, add the joined tuple to the result.
 - };
- 2) The cost of Hash-join is.

$$3(\text{Blocks of } r + \text{blocks of } s)$$
- 3) Some of the other operations in query evaluation are:
 - Duplicate elimination
 - Projection
 - Aggregate functions.
 - Set operations.

Check Your Progress 3

- 1) Methods used for evaluation of expressions:
 - (a) Materialisation
 - (b) Pipelining
- 2) Cost based optimisation consists of the following steps:
 - (a) Generating logically equivalent expressions using equivalence rules
 - (b) Generate alternative query plans.
 - (c) Choose the cheapest plan based on the estimated cost.
- 3) The evaluation plan defines exactly what algorithms are to be used for each operation and the manner in which the operations are coordinated.

“APPENDIX-A”

Equivalence Rules

- 1) The conjunctive selection operations can be equated to a sequence of individual selections. It can be represented as:

$$\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$$

- 2) The selection operations are commutative, that is,

$$\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$$

- 3) Only the last of the sequence of projection operations is needed, the others can be omitted.

$$\pi_{\text{attriblist1}}(\pi_{\text{attriblist2}}(\pi_{\text{attriblist3}} \dots (E) \dots) = \pi_{\text{attriblist1}}(E)$$

- 4) The selection operations can be combined with Cartesian products and theta join operations.

$$\sigma_{\theta_1}(E_1 \times E_2) = E_1 \bowtie_{\theta_1} E_2$$

and

$$\sigma_{\theta_2}(E_1 \bowtie_{\theta_1} E_2) = E_1 \bowtie_{\theta_2 \wedge \theta_1} E_2$$

- 5) The theta-join operations and natural joins are commutative.

$$E_1 \bowtie_{\theta} E_2 = E_2 \bowtie_{\theta} E_1$$

- 6) The Natural join operations are associative. Theta joins are also associative but with the proper distribution of joining conditions:

$$(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$$

- 7) The selection operation distributes over the theta join operation under conditions when all the attributes in the selection predicate involve only the attributes of one of the expressions (E_1) being joined.

$$\sigma_{\theta_1}(E_1 \bowtie_{\theta} E_2) = (\sigma_{\theta_1}(E_1)) \bowtie_{\theta} E_2$$

- 8) The projections operation distributes over the theta join operation with only those attributes, which are present in that relation.

$$\pi_{\text{attriblist1} \cup \text{attriblist2}}(E_1 \bowtie_{\theta} E_2) = (\pi_{\text{attriblist1}}(E_1) \bowtie_{\theta} \pi_{\text{attriblist2}}(E_2))$$

- 9) The set operations of union and intersection are commutative. But set difference is not commutative.

$$E_1 \cup E_2 = E_2 \cup E_1 \text{ and similarly for the intersection.}$$

- 10) Set union and intersection operations are also associative.

$$(E_1 \cup E_2) \cup E_3 = E_1 \cup (E_2 \cup E_3) \text{ and similarly for intersection.}$$

- 11) The selection operation can be distributed over the union, intersection, and set-differences operations.

$$\sigma_{\theta_1}(E_1 - E_2) = ((\sigma_{\theta_1}(E_1) - (\sigma_{\theta_1}(E_2)))$$

- 12) The projection operation can be distributed over the union.

$$\pi_{\text{attriblist1}}(E_1 \cup E_2) = \pi_{\text{attriblist1}}(E_1) \cup \pi_{\text{attriblist1}}(E_2)$$

UNIT 13 OBJECT-ORIENTED DATABASE

Structure	Page No.
13.0 Introduction	
13.1 Objectives	
13.2 Why Object-Oriented Database?	
13.2.1 Limitations of Relational Databases	
13.2.2 The Need for Object-Oriented Databases	
13.3 Object-Relational Database Systems	
13.3.1 Complex Data Types	
13.3.2 Types and Inheritance in SQL	
13.3.3 Additional Data Types of OOP in SQL	
13.3.4 Object Identity and Reference Type Using SQL	
13.4 Object-Oriented Database Systems	
13.4.1 Object Model	
13.4.2 Object Definition Language	
13.4.3 Object Query Language	
13.5 OODBMS Vs Object-Relational Database	
13.6 Summary	
13.7 Solutions/Answers	

13.0 INTRODUCTION

Object-oriented software development methodologies have become very popular in the development of software systems. Database applications are the backbone of most of these commercial business software developments. Therefore, it is but natural that object technologies also have their impact on database applications. Database models are being enhanced in computer systems for developing complex applications. For example, a true hierarchical data representation like a generalisation hierarchy scheme in a rational database would require a number of tables but could be a very natural representation for an object-oriented system. Thus, object-oriented technologies have found their way into database technologies. The present-day commercial RDBMS supports the features of object orientation.

This unit introduces various features of object-oriented databases. In this unit, we shall discuss the need for object-oriented databases, the complex types used in object-oriented databases, how these may be supported by inheritance, etc. In addition, we also define object definition language (ODL) and object manipulation language (OML). We shall introduce the object-oriented and object-relational databases.

13.1 OBJECTIVES

After going through this unit, you should be able to:

- define the need for object-oriented databases;
- explain the concepts of complex data types;
- use SQL to define object-oriented concepts;
- familiarise yourself with object definition and query languages, and
- define object-relational and object-oriented databases.

13.2 WHY OBJECT-ORIENTED DATABASE?

An object-oriented database is used for complex data types. Such database applications require complex interrelationships among object hierarchies to be represented in database systems. These interrelationships are difficult to implement in relational systems. Let us discuss the need for object-oriented systems in advanced applications in more detail. However, first, let us discuss the weakness of relational database systems.

13.2.1 Limitations of Relational Databases

Relational database technology was not able to handle complex application systems such as Computer-Aided Design (CAD), Computer Aided Manufacturing (CAM), Computer Integrated Manufacturing (CIM), Computer Aided Software Engineering (CASE) etc. The limitation for relational databases is that they have been designed to represent entities and relationships in the form of two-dimensional tables. Any complex interrelationship, like multi-valued attributes or composite attributes, may result in the decomposition of a table into several tables; similarly, complex interrelationships result in a number of tables being created. Thus, the main asset of relational databases, viz., its simplicity for such applications, is also one of its weaknesses in the case of complex applications.

The data domains in a relational system can be represented in relational databases as standard data types defined in the SQL. However, the relational model does not allow extending these data types or creating the user's own data types. Thus, limiting the types of data that may be represented using relational databases.

Another major weakness of the RDMS is that concepts like inheritance/hierarchy need to be represented with a series of tables with the required referential constraint. Thus, they are not very natural for objects requiring inheritance or hierarchy.

However, one must remember that relational databases have proved to be commercially successful for text-based applications and have lots of standard features, including security, reliability and easy access. Thus, even though they may not be a very natural choice for certain applications, yet their advantages are far too many. Thus, many commercial DBMS products are basically relational but also support object-oriented concepts.

13.2.2 The Need for Object-Oriented Databases

As discussed in the earlier section, relational database management systems have certain limitations. But how can we overcome such limitations? Let us discuss some of the basic issues with respect to object-oriented databases.

The objects may be complex, or they may consist of low-level objects (for example, a window object may consist of many simpler objects like a menu bar, scroll bar, etc.). However, to represent the data of these complex objects through relational database models you would require many tables – at least one each for each inherited class and a table for the base class. To ensure that these tables operate correctly, you would need to set up referential integrity constraints as well. On the other hand, object-oriented models would represent such a system very naturally through, an inheritance hierarchy. Thus, it is a very natural choice for such complex objects.

Consider a situation where you want to design a class (say an address class, which includes address lines, City, State and Pin code); the advantage of object-oriented

database management for such situations would be that they allow the representation of not only the structure but also the operation on newer user-defined database type such as finding the Address with similar Pin code. Thus, object-oriented database technologies are ideal for implementing such systems that support complex inherited objects, user defined data types (that define operations including the operations to support inheritance and polymorphism).

Another major reason for the need of object-oriented database system would be the seamless integration of this database technology with object-oriented applications. Software design is now mostly based on object-oriented technologies. Thus, object-oriented databases may provide a seamless interface for combining the two technologies.

Object-oriented databases are also required to manage complex, highly interrelated information. They provide solutions in the most natural and easy way that is closer to our understanding of the system. **Michael Brodie** related object-oriented systems to the human conceptualisation of a problem domain, which enhances communication among the system designers, domain experts, and the system end users.

The concept of an object-oriented database was introduced in the late 1970s, however, it became significant only in the early 1980s. The initial commercial product offerings appeared in the late 1980s. Some of the popular object-oriented database management systems were Objectivity/DB (developed by Objectivity, Inc.), VERSANT (developed by Versant Object Technology Corp.), Cache, ZODB, etc. An object-oriented database can be used in application areas such as e-commerce, engineering product data management, securities, medicine, etc.

Figure 1 traces the evolution of object-oriented databases. *Figure 2* highlights the strengths of object-oriented programming and relational database technologies. An object-oriented database system needs to capture the features from both these worlds. Some of the major concerns of object-oriented database technologies include access optimisation, integrity enforcement, archive, backup and recovery operations etc.

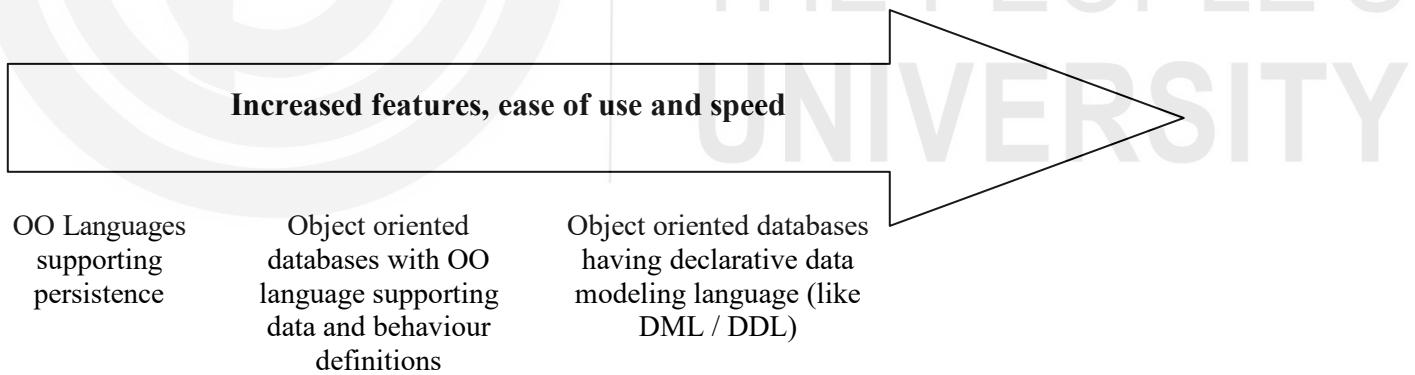


Figure 1: The evolution of object-oriented systems

Now, the question is, how does one implement an Object-oriented database system? As shown in *Figure 2* an object-oriented database system needs to include the features of object-oriented programming and relational database systems. Thus, the two most natural ways of implementing them will be either to extend the concept of object-oriented programming to include database features (OODBMS) or extend the relational database technology to include object-oriented features (Object Relational Database Systems). Let us discuss these two viz., the object relational and object-oriented database systems in more detail in the subsequent sections.

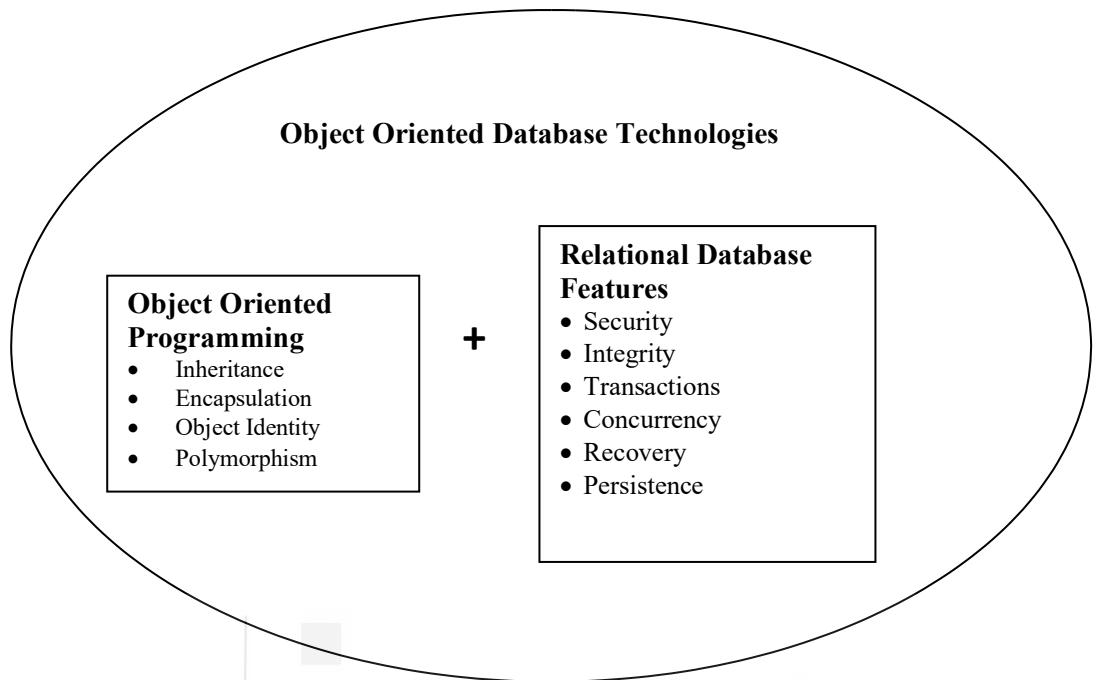


Figure 2: Makeup of an Object-Oriented Database System

13.3 OBJECT-RELATIONAL DATABASE SYSTEMS

Object-Relational Database Systems are the relational database systems that have been enhanced to include the features of object-oriented paradigm. This section provides details on how these newer features have been implemented in the SQL. Some of the basic object-oriented concepts that have been discussed in this section in the context of their inclusion into SQL standards include, the complex types, inheritance and object identity and reference types.

13.3.1 Complex Data Types

In the previous section, we have used the term complex data types without defining it. Let us explain this with the help of a simple example. Consider a composite attribute *Address*. The Address of a person in an RDBMS can be represented using the following:

- House-no and apartment
- Locality
- City
- State
- Pin-code

When using RDBMS, such information either needs to be represented as separate attributes, as shown above, or just one string separated by comma or semicolon. The second approach is very inflexible, as it would require complex string related operations for extracting information. It also hides the details of an address; thus, it is not suitable.

If you represent the attributes of the *Address* as separate attributes, then the problem would be with respect to writing queries. For example, if you need to find the address of a person, you need to specify all the attributes that you have created for the Address

viz., House-no, Locality.... etc. The question is - Is there any better way of representing such information using a single field? If there is such a mode of representation, then that representation should permit distinguishing each element of the *Address*. The following may be one such possible solution:

```
CREATE TYPE Address AS      (
    Addressline1      Char (20)
    Addressline2      Char (20)
    City              Char (12)
    State             Char (15)
    Pincode           Char (6)
) ;
```

Thus, *Address* is now a new type that can be used while creating a database system schema as:

```
CREATE TABLE STUDENT      (
    name            Char (25)
    address         Address
    phone           Char (12)
    programme       Char (5)
    dob             Date
) ;
```

But what are the advantages of such definition?

Consider the following queries:

Find the name and address of the students enrolled in the PGDCA programme.

```
SELECT      name, address
FROM        student
WHERE       programme = 'PGDCA' ;
```

Please note that the attribute ‘address’, although composite, is used as a single attribute in the query. But can you also refer to individual components of this attribute?

Find the name and address of all the PGDCA students of Mumbai.

```
SELECT      name, address
FROM        student
WHERE       programme = 'MCA' AND address.city='Mumbai' ;
```

Thus, such definitions allow you to handle a composite attribute as a single attribute with a user-defined type. You can also refer to any of the components of this attribute without any problems, so the data definition of the components of the composite attribute is still intact.

Complex data types also allow you to model a table with multi-valued attributes, which would require a new table in a relational database design. For example, a library database system would require representing the following information of a book.

- ISBN number
- Book title
- Authors
- Published by
- Subject areas of the book.

Clearly, in the table above, authors and subject areas are multi-valued attributes. You

need to design two relational database tables for these attributes – Author (ISBNnumber, author) and Area (ISBNnumber, subject area). (Please note that this design does not consider the author's position in the list of authors).

Although this database solves the immediate problem, yet it is a complex design. An object-oriented database system may solve this problem. This is explained in the next section.

13.3.2 Types and Inheritances in SQL

In the previous sub-section, we discussed the data type – *Address*. It is a good example of a structured type. In this section, let us give more examples for such types, using SQL. Consider the attribute:

- Name – that includes given name, middle name and surname.
- Address – that includes address details, city, state and pin code.

These types can be defined using SQL extensions, as given below:

```
CREATE TYPE Name AS (
    Given_name      Char (20),
    Middle_name     Char (15),
    Sur_name        Char (20)
)
FINAL
```

This type/class cannot be inherited further due to the keyword FINAL.

```
CREATE TYPE Address AS (
    Add_det      Char (20),
    City         Char (20),
    State        Char (20),
    Pincode      Char (6)
)
NOT FINAL
```

You can use this class to create inherited classes, like *Home_Address* and *Office_Address*, as this type/class is NOT FINAL.

The FINAL and NOT FINAL keywords have the same meaning as you learned in JAVA, i.e., a FINAL class cannot be inherited further.

These types can now be used to create a student class, which has data members and methods that work on objects of the student class, as follows:

```
CREATE TYPE Student AS (
    name          Name,
    address       Address,
    dob           Date
)
NOT FINAL
METHOD ageinyears (givendate Date)
    RETURN INTERVAL YEAR;
```

The method can be implemented separately using the following SQL Commands:

```

CREATE INSTANCE METHOD (givendate Date)
    RETURN INTERVAL YEAR
FOR Student
begin
    Return (givendate - self.dob);
end

```

This method computes the age on a given date. Please note that Date is a data type of SQL. FOR is the looping construct and will result in the execution of this method for every student object's instance.

The possibility of using constructors also exists, but a detailed discussion on that is beyond the scope of this unit.

Type Inheritance

In the present standard of SQL, you can define inheritance. Let us explain this with the help of an example.

Consider a type *University_person* defined as:

```

CREATE TYPE      University_person AS (
    name        Name,
    address     Address
)

```

Now, this type can be inherited by the *Staff* type or the *Student* type. For example, the *Student* type, if inherited from the class given above, would be:

```

CREATE TYPE      Student
UNDER           University_person (
    programme   Char(10),
    dob         Date
)

```

Similarly, you can create a sub-class for the *Staff* of the University as:

```

CREATE TYPE      Staff
UNDER           University_person (
    designation Char(10),
    basic_salary Number(7)
)

```

Notice that both the inherited types shown above inherit the *name* and *address* attributes from the type *University_person*. Methods can also be inherited in a similar way; however, they can be overridden if the need arises.

Table Inheritance

The concept of table inheritance has evolved to incorporate implementation of generalisation/ specialisation hierarchy of an E-R diagram. SQL allows inheritance of tables. Once a new type is declared, it could be used in the process of creation of new tables with the usage of keyword “OF”. Let us explain this with the help of an example. Consider the classes *University_person*, *Staff* and *Student*, as we have defined in the previous sub-section. You can create the table for the type *University_person* as:

```
CREATE TABLE university_members OF University_person;
```

The table inheritance would allow us to create sub-tables for such tables as:

```
CREATE TABLE student_list OF Student  
    UNDER university_members;
```

Similarly, you can create a table for the staff as:

```
CREATE TABLE staff OF Staff  
    UNDER university_members;
```

Please note the following points for table inheritance:

- The type that is associated with the sub-table must be the sub-type of the type of the parent table. This is a major requirement for table inheritance.
- All the attributes of the parent table – (university_members in our case) should be present in the inherited tables.
- Also, the three tables may be handled separately. However, any record present in the inherited tables is also implicitly present in the base table. For example, any record inserted in the student_list table will be implicitly present in university_members tables.
- A query on the parent table (such as university_members) would find the records from the parent table and all the inherited tables (in our case all three tables). However, the attributes of the result table would be the same as the attributes of the parent table.
- You can restrict your query to only the parent table by using the keyword ONLY. For example,

```
SELECT name FROM university_members ONLY;
```

13.3.3 Additional Data Types of OOP in SQL

The object-oriented/relational database must support the data types that allow multi-valued attributes to be represented easily. Two such data types that exist in SQL are:

- Arrays – stores information in an order and
- Multisets – stores information in an unordered set.

Let us explain this with the help of an example of a book database as introduced in the section 13.3.1. A Book type can be represented using SQL as:

```
CREATE TYPE Book AS (  
    ISBNNO          Char (14),  
    BOOK_TITLE      Char (25),  
    AUTHORS          Char (25) ARRAY [5],  
    PUBLISHED_BY    Char (20),  
    KEYWORDS         Char (10) MULTISET  
) ;
```

Please note, the use of the type ARRAY. Arrays not only allow authors to be represented but also allow the sequencing of the authors' names. Multiset allows a number of keywords without any ordering imposed on them.

But how can you enter data and query such data types? The following SQL commands would help in defining such a situation. But first, you need to create a table:

```
CREATE TABLE library OF Book;  
INSERT INTO library VALUES('008-124476-x', 'Database  
Systems', ARRAY ['Silberschatz', 'Elmasri'],  
'XYZ Publication', MULTISET ['Database',
```

```
'Relational', 'Object Oriented']) ;
```

The command above would insert information on a hypothetical book into the database. Let us now write a few queries on this database:

Find the list of books related to area Object Oriented:

```
SELECT ISBNNO, BOOK_TITLE  
FROM library  
WHERE 'Object Oriented' IN (UNNEST (KEYWORDS)) ;
```

Find the first author of each book:

```
SELECT ISBNNO, BOOK_TITLE, AUTHORS[1]  
FROM library ;
```

You can create many such queries. A detailed discussion on this can be found in the latest SQL standards and is beyond the scope of this unit.

13.3.4 Object Identity and Reference Type Using SQL

Till now, we have created the tables, but what about the situation when we have attributes that draw a reference to another attribute in the same table? This is a referential constraint. Thus, an object-relational database system should address the following two concerns:

- In relational databases, foreign keys are used to link the attributes in two different relations. Can such keys be used in an object-relational database?
- How will you identify the object that is being referenced?

The following example will address the concerns stated above.

Example: A library purchases books. Each book is given a unique book number called the catalogue number. The library maintains a procurement table, which can be created using the following SQL command:

```
CREATE TABLE procurement (  
    CATALOGUE_NO     CHAR (5),  
    ISBNNO REF (Book) SCOPE (library)  
) ;
```

The SQL statement given above would create a procurement table, which would assign two basic information to a newly purchased book – first, it will give the book a unique CATALOGUE_NO, and second, it will link this book to a specific record in the library table through an ISBNNO.

To insert a new book in this table, you must first create a Book object using the ISBNNO of this book. Assuming that such an object already exists, then you may use the following command to add a book to the procurement table:

```
INSERT INTO procurement  
VALUES ('98765', NULL) ;
```

This command will add a new book to the procurement table having CATALOGUE_NO as '98765' and no reference to the ISBN number. The link to the ISBN number record will be created using the following SQL command:

```
UPDATE procurement
```

```
SET ISBNNO = (SELECT book_id  
FROM library  
WHERE ISBNNO = '83-7758-476-6')  
WHERE CATALOGUE_NO = '98765';
```

Please note that in the query above, the sub-query generates the object identifier (book_id) for the ISBNNO of the book whose accession number is 98765. It then sets the reference for the desired record in the procurement.

This is a slightly complex procedure, and several other mechanisms exist to perform this operation. You can refer to them in the further readings.

☛ Check Your Progress – 1

- 1) What is the need for object-oriented databases?

.....
.....
.....

- 2) How will you represent a complex data type?

.....
.....
.....

- 3) Represent an address using SQL that has a method for locating pin-code information.

.....
.....
.....

- 4) Create a table using the type created in question 3 above.

.....
.....
.....

- 5) How can you establish a relationship with multiple tables?

.....
.....
.....

13.4 OBJECT-ORIENTED DATABASE SYSTEMS

Object-oriented database systems are applying object-oriented concepts into database system models to create an object-oriented database model. This section describes the concepts of the object model, followed by a discussion on object definition and object manipulation languages that are derived from SQL.

13.4.1 Object Model

Object Data Management Group (ODMG) has designed the object model for the object-oriented database management system. The Object Definition Language (ODL) and Object Manipulation Language (OML) are based on this object model. Let us briefly define the concepts and terminology related to the object model.

Objects and Literal: These are the basic building elements of the object model. An object has the following four characteristics:

- A unique identifier
- A name
- A lifetime, defining whether it is persistent or not, and
- A structure that may be created using a type constructor. The structure in OODBMS can be classified as atomic or collection objects (like Set, List, Array, etc.).

A literal does not have an identifier but has a value that may be constant. The structure of a literal does not change. Literals can be atomic, such that they correspond to basic data types like int, short, long, float, etc. or structured literals (for example, current date, time, etc.) or collection literal defining values for some collection object.

Interface: Interfaces define the operations that can be inherited by a user-defined object. Interfaces are non-instantiable. All objects inherit basic operations (like copy object, delete object) from the interface of Objects. A collection object inherits operations – such as an operation to determine an empty collection – from the basic collection interface.

Atomic Objects: An atomic object is an object that is not of a collection type. They are user-defined objects that are specified using class keywords. The properties of an atomic object can be defined by its attributes and relationships. An example is the book object given in the next subsection. **Please note** here that a *class* is instantiable.

Inheritance: The interfaces specify the abstract operations that can be inherited by classes. This is called behavioural inheritance and is represented using the “ : ” symbol. Sub-classes can inherit the state and behaviour of super-class(s) using the keyword EXTENDS.

Extents: An extent of an object contains all the persistent objects of that class. A class having an extent can have a key.

In the following section, we shall discuss the use of the ODL and OML to implement object models.

13.4.2 Object Definition Language

Object Definition Language (ODL) is a standard language on the same lines as the DDL of SQL, that is used to represent the structure of an object-oriented database. It uses unique object identity (OID) for each object such as library item, student, account, fees, inventory etc. In this language, objects are treated as records. Any class in the design process has three properties that are attribute, relationship, and methods. A class in ODL is described using the following syntax:

```
class <name>
{
<list of properties>
};
```

Here, *class* is a keyword, and the properties may be attributes, methods or relationships. The attributes defined in ODL specify the features of an object. It could be simple, enumerated, structured or complex type.

```

class Book
{
    attribute string ISBNNO;
    attribute string BOOKTITLE;
    attribute enum CATEGORY
        {text,reference,journal} BOOKTYPE;
    attribute struct AUTHORS
        {string fauthor, string sauthor,
         string tauthor} AUTHORLIST;
};

```

Please note that, in this case, we have defined authors as a structure and a new field on Book type as an enum.

These books need to be issued to the students. For that, you need to specify a relationship. The relationship defined in ODL specifies the method of connecting one object to another. You can specify the relationship by using the keyword “relationship”. For example, to connect a student object with a book object, you need to specify the relationship in the student class as:

```
relationship set <Book> receives
```

Here, for each object of the class student, there is a reference to the book object and the set of references is called *receives*.

But if we want to access the student based on the book, then the “inverse relationship” could be specified as

```
relationship set <Student> receivedby
```

You can specify the connection between the relationship *receives* and *receivedby* by using the keyword “inverse” in each declaration. If the relationship is in a different class, it is referred to by the relationship name followed by a scope resolution operator (::) and the name of the other relationship.

The relationship could be specified as:

```

class Book
{
    attribute string ISBNNO;
    attribute string BOOKTITLE;
    attribute integer PRICE;
    attribute string PUBLISHEDBY;
    attribute enum CATEGORY
        {text,reference} BOOKTYPE;
    attribute struct AUTHORS
        {string fauthor
         string sauthor
         string tauthor} AUTHORLIST;
    relationship set <Student> receivedby
        inverse Student::receives;
    relationship set <Supplier> suppliedby
        inverse Supplier::supplies;
};
class Student
{
    attribute string ENROLMENT_NO;

```

```

        attribute string NAME;
        attribute integer MARKS;
        attribute string COURSE;
        relationship set <Book> receives
            inverse Book::receivedby;
    };
class Supplier
{
    attribute string SUPPLIER_ID;
    attribute string SUPPLIER_NAME;
    attribute string SUPPLIER_ADDRESS;
    attribute string SUPPLIER_CITY;
    relationship set <Book> supplies
        inverse Book::suppliedby;
};

```

Methods could be specified with the classes along with input/output types. These declarations are called “signatures”. These method parameters could be *in*, *out* or *inout*. Here, the ‘*in*’ parameter is passed by value, whereas the ‘*out*’ or ‘*inout*’ parameters are passed by reference. Exceptions could also be associated with these methods.

```

class Student
{
    attribute string ENROLMENT_NO;
    attribute string NAME;
    attribute string st_address;
    relationship set <Book> receives
        inverse Book::receivedby;
    void findcity(in set<string>,out set<string>)
        raises(notfoundcity);
};

```

In the method *findcity*, the name of the city is passed with the objective to find the name of the students who belong to that specific city. In case blank is passed as a parameter for city name, then the exception *notfoundcity* is raised. The implementation of this method can be done separately.

The ODL could be atomic type or class names. The basic type uses many class constructors such as set, bag, list, array, dictionary and structure. We have shown the use of some in the example above. You can refer to the further readings for more detail on these.

Inheritance is implemented in ODL using subclasses with the keyword “extends”.

```

class Journal extends Book
{
    attribute string VOLUME;
    attribute string emailauthor1;
    attribute string emailauthor2;
};

```

Multiple inheritance is implemented by using extends separated by a colon (:). If there is a class Fee containing fee details, then multiple inheritance could be shown as:

```

class StudentFeeDetail extends Student:Fee
{
    void deposit(in set <float>, out set <float>)

```

```

        raises(refundToBeDone)
    };
```

Like the difference between relation schema and relation instance, ODL uses the class and its extent (set of existing objects). The objects are declared with the keyword “extent”.

```

class Student (extent firstStudent)
{
    attribute string ENROLMENT_NO;
    attribute string NAME;
    .....
};
```

It is not necessary in the case of ODL to define keys for a class. But if one or more attributes have to be declared as keys, then it may be done with the declaration of a key for a class with the keyword “key”.

```

class student (extent firstStudent key ENROLMENT_NO)
{
    attribute string ENROLMENT_NO;
    attribute string NAME;
    .....
};
```

Assuming that the ENROLMENT_NO and ACCESSION_NO form a key for the issue table, then:

```

class Issue(extent thisMonthIssue key
            (ENROLMENT_NO,ACCESSION_NO))
{
    attribute string ENROLMENT_NO;
    attribute string ACCESSION_NO;
    .....
};
```

The major considerations while converting ODL designs into relational designs are as follows:

- It is not essential to declare keys for a class in ODL, but in Relational design new attributes have to be created as a key.
- Attributes in ODL could be declared as non-atomic, whereas in Relational design they have to be converted into atomic attributes.
- Methods could be part of the design in ODL, but they cannot be directly converted into a relational schema, as they are not the property of a relational schema.
- Relationships are defined in inverse pairs for ODL, but in the case of relational design only one pair is defined.

For example, for the book class schema, the relation is:

```
Book (ISBNNO, TITLE, CATEGORY, fauthor, sauthor, tauthor)
```

ODL has been created with the features required to create an object-oriented database in OODBMS. You can refer to the further readings for more details on it.

13.4.3 Object Query Language

Object Query Language (OQL) is a standard query language that takes high-level declarative programming of SQL and object-oriented features of OOPs. Let us explain it with the help of examples.

Find the list of authors for the book titled “OODBMS”.

```
SELECT b.AUTHORS  
FROM Book b  
WHERE BOOK_TITLE="OODBMS" ;
```

Display the title of the book which has been issued to the student whose name is Anand.

```
SELECT BOOK_TITLE  
FROM Book b, Student s  
WHERE s.NAME ="Anand" ;
```

This query can also be written by using a relationship as:

```
SELECT BOOK_TITLE  
FROM Book b  
WHERE b.receivedby.NAME ="Anand" ;
```

In the previous case, the query creates a bag of strings, but when the keyword DISTINCT is used, the query returns a set.

```
SELECT DISTINCT BOOK_TITLE  
FROM Book b  
WHERE b.receivedby.NAME ="Anand" ;
```

When you add the ORDER BY clause to return a sorted list.

```
SELECT BOOK_TITLE  
FROM Book b  
WHERE b.receivedby.NAME ="Anand"  
ORDER BY b.CATEGORY ;
```

Aggregate operators like SUM, AVG, COUNT, MAX, MIN could be used in OQL. If you want to compute the maximum marks obtained by any student, then the OQL command is

```
Max(SELECT s.MARKS FROM Student s);
```

Group By and Having clauses can also be used; however, you may refer to further readings for details on them.

Union, intersection and difference operators are applied to set or bag type with the keywords UNION, INTERSECT and EXCEPT. If you want to display the details of suppliers from PATNA and SURAT, then the OQL command is:

```
(SELECT DISTINCT su  
    FROM Supplier su  
    WHERE su.SUPPLIER_CITY="PATNA")  
UNION  
(SELECT DISTINCT su  
    FROM Supplier su  
    WHERE su.SUPPLIER_CITY="SURAT");
```

The result of the OQL expression could be assigned to host language variables. If costlyBooks is a set <Book> variable to store the list of books whose price is more than Rs. 500, then:

```

costlyBooks = SELECT DISTINCT b
    FROM Book b
    WHERE b.PRICE > 500

```

In this section, you have been introduced to OQL. You can refer to further readings for more details on OQL.

Check Your Progress – 2

- 1) Create a class staff using ODL that also references the Book class given in section 13.4.
-
.....

- 2) What modifications would be needed in the Book class because of the table created by the above query?
-
.....

- 3) Find the list of books that have been issued to “Shashi”.
-
.....

13.5 OODBMS VERSUS OBJECT RELATIONAL DATABASE

An object-oriented database management system is created on the basis of persistent programming paradigm, whereas an object-relational is built by creating object-oriented extensions of a relational system. In fact, both the products have clearly defined objectives. The following table shows the difference between them:

Object-Relational DBMS	Object-Oriented DBMS
The features of these DBMS include: <ul style="list-style-type: none"> • Support for complex data types • Powerful query languages support through SQL • Good protection of data against programming errors 	The features of these DBMS include: <ul style="list-style-type: none"> • Support complex data types, • Very high integration of database with the programming language, • Very good performance • But not as powerful at querying as Relational.
One of the major assets here is SQL. Although, SQL is not as powerful as a Programming Language, but it is none-the-less essentially a fourth-generation language, thus, it provides excellent protection of data from the Programming errors.	It is based on object-oriented programming languages, thus, are very strong in programming, however, any error of a data type made by a programmer may affect many users.
The relational model has a very rich foundation for query optimisation, which helps in reducing the time taken to execute a query.	These databases are still evolving in this direction. They have reasonable systems in place.

These databases make the querying as simple as in relational even, for complex data types and multimedia data.	The querying is possible but somewhat difficult to get.
Although the strength of these DBMS is SQL, it is also one of the major weaknesses from the performance point of view of in memory applications.	Some applications that are primarily run in the RAM and require a large number of database accesses with high performance may find such DBMS more suitable. This is because of rich programming interface provided by such DBMS. However, such applications may not support very strong query capabilities. A typical example of one such application is databases required for CAD.

☛ Check Your Progress – 3

State True or False.

- 1) Object-relational databases cannot represent inheritance but can represent complex data types. T F
- 2) The class extent defines the limit of a class. T F
- 3) The query language of object-oriented DBMS is stronger than object-relational databases. T F
- 4) SQL commands cannot be optimised. T F
- 5) Object-oriented DBMS supports a very high level of integration of databases with OOP. T F

13.6 SUMMARY

Object-oriented technologies are one of the most popular technologies in the present era. Object orientation has also found its way into database technologies. The object-oriented database systems allow the representation of user-defined types, including operation on these types. They also allow the representation of inheritance using the type and table inheritance. The idea here is to represent the whole range of newer types if needed. Such features help in enhancing the performance of a database application that would otherwise have many tables. SQL support these features for object-relational database systems.

The object definition languages and object query languages have been designed for the object-oriented DBMS on the same lines as that of SQL. These languages try to simplify various object-related representations using OODBMS.

The object-relational and object-oriented databases do not compete with each other but have different kinds of application areas. For example, relational and object-relational DBMS are most suited for simple transaction management systems, while OODBMS may find applications with e-commerce, CAD and other similar complex applications.

13.7 SOLUTIONS/ANSWERS

Check Your Progress 1

- 1) The object-oriented databases are need for:
- Representing complex types.
 - Representing inheritance, polymorphism
 - Representing highly interrelated information
 - Providing object-oriented solution to databases bringing them closer to OOP.

- 2) Primarily by representing it as a single attribute. All its components should also be referenced separately.

3)

```
CREATE TYPE Addrtype AS
(
    houseNo      CHAR(8),
    street       CHAR(10),
    colony        CHAR(10),
    city          CHAR(8),
    state         CHAR(8),
    pincode       CHAR(6),
);

METHOD pin() RETURNS CHAR(6);
CREATE METHOD pin() RETURNS CHAR(6);
FOR Addrtype
BEGIN
    Return pincode;
END
```

- 4) CREATE TABLE addresswithpin OF Addrtype
(
 REF IS addressid,
 PRIMARY KEY
 (street,colony,city,pincode)
) ;

- 5) The relationship can be established with multiple tables by specifying the keyword “SCOPE”. For example:

```
CREATE TABLE mylibrary
(
    mybook REF(Book) SCOPE library;
    myStudent REF(Student) SCOPE student;
    mySupplier REF(Supplier) SCOPE supplier;
)
```

Check Your Progress 2

1)

```
class Staff
{
    attribute string STAFF_ID;
```

```
attribute string STAFF_NAME;
attribute string DESIGNATION;
relationship set <Book> issues
    inverse Book::issuedto;
};
```

- 2) The Book class needs to represent the relationship that is with the Staff class.

This would be added to it by using the following commands:

```
RELATIONSHIP SET < Staff > issuedto
    INVERSE :: issues Staff
```

- 3) SELECT DISTINCT b.TITLE
FROM BOOK b
WHERE b.issuedto.NAME = "Shashi"

Check Your Progress 3

- 1) False 2) False 3) False 4) False 5) True



UNIT 14 DATA WAREHOUSING AND DATA MINING

Structure

- 14.0 Introduction
- 14.1 Objectives
- 14.2 What Is Data Warehousing?
- 14.3 Basic Components of a Data Warehouse
 - 14.3.1 The Data Sources
 - 14.3.2 Data Extraction, Transformation and Loading (ETL)
- 14.4 Multidimensional Data Model of a Data Warehouse
- 14.5 Data Mining Technology
- 14.6 Classification
 - 14.6.1 Classification Using Distance (K-Nearest Neighbour)
 - 14.6.2 Decision Tree
- 14.7 Clustering
- 14.8 Association Rule Mining
- 14.9 Applications of Data Mining
- 14.10 Summary
- 14.11 Solutions/Answers
- 14.12 Further Readings

14.0 INTRODUCTION

With the advancement of communication technology, a large amount of data is generated and collected by various organisations. This large data can be used for making meaningful decisions, which can be attributed to discovering useful knowledge and patterns from their existing data. One of the ways of storing vast reservoirs of data is data warehousing. Data warehouses provide superior capabilities for data storage, processing, and responsiveness to analytical queries compared to transaction-oriented databases. In decision-making applications, users need to access a larger volume of data very rapidly – much faster than what can be conveniently handled by traditional database systems. Often, such data is extracted from multiple operational databases. The data of a data warehouse can be used by a data mining application. Data mining is an interdisciplinary field that takes its approach from different areas like databases, statistics, artificial intelligence and data structures to extract hidden knowledge from large volumes of data. The data mining concept is used by the research community and the industry for various kinds of data analysis.

This unit aims to introduce you to some of the fundamental techniques used in data warehousing and data mining. This unit introduces only those data warehousing concepts which relate to structured data.

14.1 OBJECTIVES

After going through this unit, you should be able to:

- Illustrate the concepts of a data warehouse;
- discuss data warehousing schemas;
- identify the multi-dimensional data modelling of a data warehouse;
- explain the purpose of data mining and its applications;
- illustrate classification and clustering approaches of data mining;
- explain how association rules can be identified in data mining.

14.2 WHAT IS DATA WAREHOUSING?

Let us first try to answer the question: What is a data warehouse? A simple answer could be: A data warehouse is a tool that manages data *after and outside* of operational systems. Thus, it is not a replacement for the operational systems but is a major tool that acquires data from the operational systems. Data warehousing technology has evolved in business applications for the process of strategic decision-making. Data warehouses, sometimes, may be considered the key components of the IT strategy and architecture of an organisation.

The formal definition of the data warehouse was given by **W.H. Inman**: “A Data warehouse is a collection of data, which is (i) *subject-oriented*, (ii) *integrated*, (iii) *nonvolatile* (iv) *time-variant* that supports decision-making by the management”. *Figure 1* presents some uses of data warehousing in various industries.

S.No.	Industry	Functional Areas of Use	Strategic Uses
1	Banking	Creating new schemes for loans and other banking products; helps in operations; identifies information for marketing.	Finding trends for customer service; product and service promotions; reduction of expenses.
2	Airline	Operations; marketing.	Crew assignment; aircraft maintenance plans; fare determination; analysis of route profitability; frequent - flyer program design.
3	Hospital	Operation optimisation.	Reduction of operational expenses; scheduling of resources.
4	Investment and Insurance	Insurance product development; marketing	Risk management; financial market analysis; customer tendencies analysis; portfolio management

Figure 1: Uses of Data Warehousing

A data warehouse offers the following advantages:

- It provides historical information that can be used in many different forms of analysis.
- Improves the data quality by predicting missing data.
- It can help support disaster recovery, although not alone, but with other backup resources.

One of the major advantages of a data warehouse is that it allows a large collection of historical data from many operational databases that can be analysed through one data warehouse interface. A data warehouse does not create value of its own in an organisation. However, the value can be generated by the users of the data warehouse. For example, a data warehouse of a manufacturing unit may answer some of the following questions:

- What would be the income, expenses and profit for a year?
- What would be the sales amount this month?
- Who are the vendors for a product that is to be procured?
- How much of each product is manufactured in each production unit?
- How much is to be manufactured?

- What percentage of the product is defective?
- Are customers satisfied with the quality? etc.

The data warehouse supports various business intelligence applications. Some of these may be - online analytical processing (OLAP), decision-support systems (DSS), data mining etc. We shall discuss some of these terms in more detail in the later sections.

14.3 BASIC COMPONENTS OF A DATA WAREHOUSE

A data warehouse is defined as a *subject-oriented, integrated, nonvolatile, time-variant collection*, but how can we achieve such a collection? To answer this question, let us define the basic architecture that helps a data warehouse achieve the objectives stated above. We shall also discuss various processes that are performed by these components on the data.

Figure 2 defines the basic architecture of a data warehouse. The analytical reports are not a part of the data warehouse but are one of the major business application areas including OLAP, DSS and Data Mining

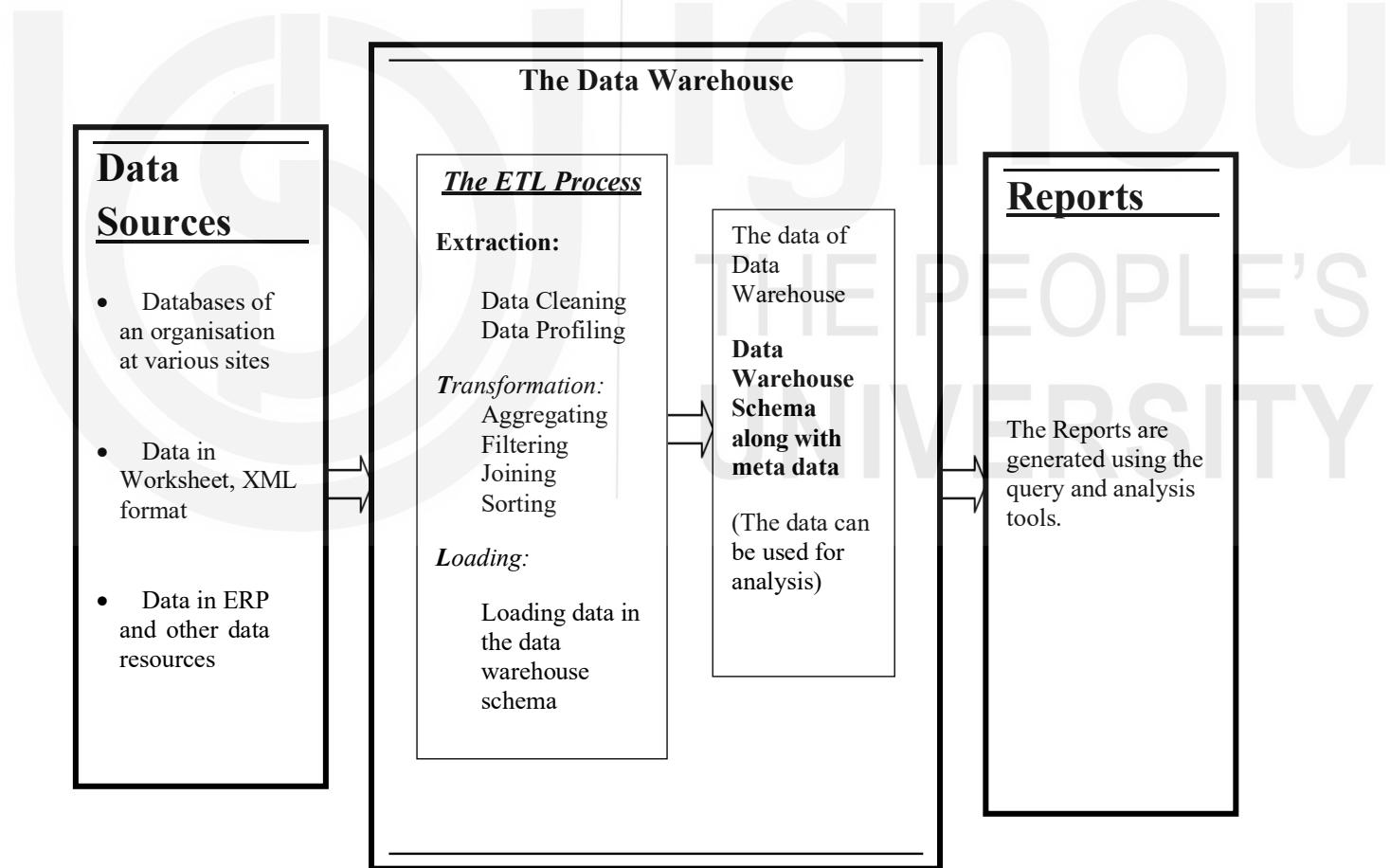


Figure 2: The Data Warehouse Architecture

14.3.1 The Data Sources

The data of the data warehouse can be obtained from many operational systems. A data warehouse interacts with the environment that provides most of the source data for the data warehouse. By the term environment, we mean traditionally developed database

systems and other applications. In a large installation, hundreds or even thousands of these database systems or files-based systems exist with plenty of redundant data.

The warehouse database obtains most of its data from such different forms of legacy systems – files and databases. Data may also be sourced from external sources as well as other organisational systems. This data needs to be integrated into the warehouse. But how do you integrate the data of these large numbers of operational systems into the data warehouse system? You need the help of ETL tools to do so. These tools capture the data that is required to be put in the data warehouse.

Data in Data Warehouse

The basic characteristics of the data in a data warehouse can be described in the following way:

- i) **Subject Oriented:** The first characteristic of the data warehouse's data is that its design and structure can be oriented to important objects of the organisation. These objects for a university data warehouse can be STUDENT, PROGRAMME, REGIONAL CENTRES etc., whereas the operational systems may be designed around applications and functions such as ADMISSION, EXAMINATION and RESULT DECLARATIONS (in the case of a University). Refer to *Figure 3*.

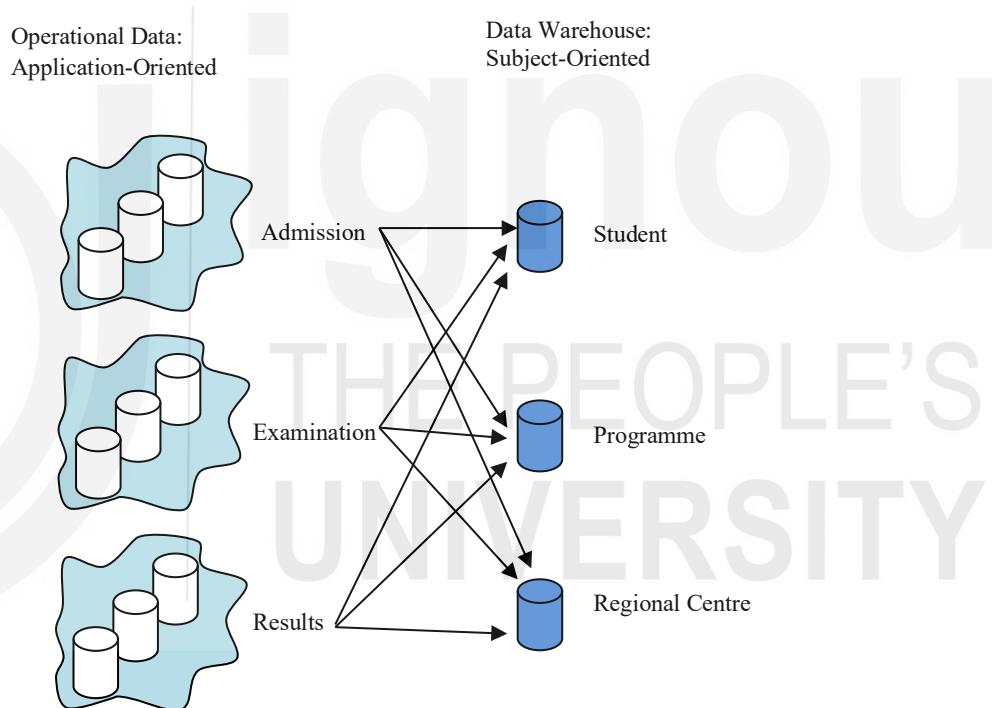


Figure 3: Operations system data vs Data warehouse data

- ii) **Integrated:** Integration means bringing together data from multiple dissimilar operational sources on the basis of an enterprise data model. A data model of a data warehouse can be a basic template that uniquely defines the organisation's key data items. Data integration requires:

- Standardising the data naming and definition: for example, “student enrolment number” can be standardised over the data of the entire university.
- Standardising of encodings: for example, the first two digits of an enrolment number represent the year of admission.
- Standardising the Measurement of variables: for example, all the units would be expressed in the metric system and all monetary details would be given in Indian Rupees.

- iii) **Time-Variant:** The third defining characteristic of the data in a data warehouse is that it is time-variant or historical in nature. The entire data in the data warehouse is/was accurate at some point in time. This is in contrast with operational data that changes over a shorter time period. *Figure 4* defines this characteristic of a data warehouse.

OPERATIONAL DATA	DATA WAREHOUSE DATA
It is the current value data	Contains a snapshot of historical data
Time span of data = 60-90 days	Time span of data = 5-10 years or more
Data can be updated in most cases	After making a snapshot of the data record cannot be updated
May or may not have a timestamp	Will always have a timestamp

Figure 4: Characteristics of data of operational data and data in a data warehouse

- iv) **Non-volatile:** Data loaded in a data warehouse is subsequently scanned and used but is not updated in the same classical ways as the operational system's data, which is updated through the transaction processing cycles.

Metadata Directory

The metadata directory component defines the repository of the information stored in the data warehouse. The metadata can be used by general users as well as data administrators. It contains the following information:

- i) the structure of the contents of the data warehouse data,
- ii) the source of the data,
- iii) the data transformation processing requirements, such that data can be passed from the legacy systems into the data warehouse database,
- iv) the process summarisation of data,
- v) the data extraction history, and
- vi) how the data needs to be extracted from the data warehouse.

14.3.2 Data Extraction, Transformation and Loading (ETL)

The first step in data warehousing is to perform data extraction, transformation, and loading of data into the data warehouse. This is called ETL, which is Extraction, Transformation, and Loading. ETL refers to the methods involved in accessing and manipulating data available in various sources and loading it into a target data warehouse.

What happens during the ETL Process?

The following are the sub-processes of the ETL process of the data warehouse:

Data Extraction: The ETL is a three-stage process. During the *Extraction* phase, the desired data is identified and extracted from many different sources. These sources may be different databases or non-databases. The process of extraction sometimes involves some basic transformation. For example, if the data is being extracted from two Sales databases where the sales in one of the databases are in Dollars and the other in Rupees, then a simple transformation would be required in the data. The size of the extracted data may vary from hundreds of kilobytes to hundreds of gigabytes, depending on the data sources and business systems.

The *extraction* process involves data cleansing and data profiling. Data cleansing can be defined as the process of removal of inconsistencies in the data. For example, the

state name may be written in many ways, and they can be misspelt too. For example, the state Uttar Pradesh may be written as U.P., UP, Uttar Pradesh, Utter Pradesh etc. The cleansing process may try to correct the spellings as well as resolve such inconsistencies. But how does the cleansing process do that? One simple way may be to create a Database of the States with possible fuzzy matching algorithms that may map various variants into one state name. Thus cleansing the data to a great extent.

Data profiling involves creating the necessary data from the point of view of a data warehouse application. Another concern here is to eliminate duplicate data. For example, an address list collected from different sources may be merged and purged to create an address profile with no duplicate data.

Data Transformation: One of the most time-consuming tasks - *data transformation* follows the extraction stage. The data transformation process includes the following:

- Use of data filters,
- Data validation against the existing data,
- Checking data duplication, and
- Information aggregation.

Transformations are useful for transforming the source data according to the requirements of the data warehouse. The process of transformation should ensure the quality of the data that needs to be loaded into the target data warehouse. Some of the common transformations are:

Filter Transformation: Filter transformations are used to filter the rows in a mapping that do not meet specific conditions. For example, the list of employees of the Sales department who made sales above Rs.50,000/- may be filtered out.

Joiner Transformation: This transformation is used to join the data of one or more different tables that may be stored in two different locations and could belong to two different sources of data that may be relational or other data formats like XML.

Aggregator Transformation: Such transformations perform aggregate calculations on the extracted data. Some such calculations may find the sum or average.

Sorting Transformation: requires creating an order in the required data based on the application requirements of the data warehouse.

Data Loading: Once the data for the data warehouse is properly extracted and transformed, it is *loaded* into a data warehouse. This process requires creation and execution of programs that perform this task. One of the key concerns here is to propagate source data updates. Sometimes, this problem is equated to the problem of maintenance of the materialised views.

When should we perform the ETL process for the data warehouse? ETL process should normally be performed at night or when the load on the operational systems is low. **Please note** that the integrity of the extracted data can be ensured by synchronising different operational applications feeding the data warehouse and the data in the data warehouse.

Check Your Progress – 1

- 1) What is a Data Warehouse?

.....

.....
.....
.....
.....
.....

2) What are the characteristics of data in a data warehouse?

.....
.....
.....
.....

3) What is ETL? What are the different transformations that are needed during the ETL process?

.....
.....
.....
.....

14.4 MULTIDIMENSIONAL DATA MODEL OF A DATA WAREHOUSE

A data warehouse is a huge collection of data. Such data may involve grouping of data on multiple attributes. For example, the enrolment data of the students at a University may be represented using a student schema such as:

Student_enrolment (year, programme, region, number)

Some data values for such schema are (Also refer to *Figure 5*, which shows this data):

- In the year 2002, BCA enrolment at Region (Regional Centre Code) RC-07 (Delhi) was 350.
- In the year 2003, BCA enrolment in Region RC-07 was 500.
- In the year 2002, MCA enrolment in all the regions was 8000.

Please note that for representing the value of a number of students, you need to refer to three attributes: year, programme and region. Each of these attributes is identified as a dimension attribute. Thus, the data of the *Student_enrolment* table can be modelled using three-dimensional attributes (year, programme, region) and a measure attribute (number). Such kind of data is referred to as multidimensional data. Thus, a data warehouse may use multidimensional matrices referred to as a data cube model. If the dimensions of the matrix are greater than three, then it is called a hypercube. *Figure 5* represents the multidimensional data of a university:

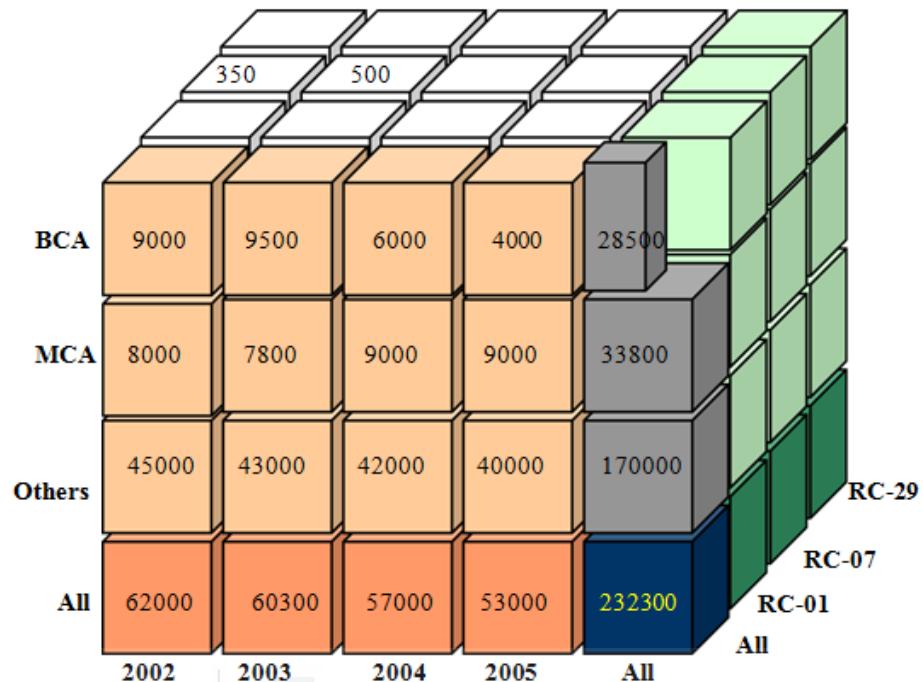


Figure 5: Sample multidimensional data

Multidimensional data may be a little difficult to analyse. Therefore, Multidimensional data may be displayed on a certain pivot, for example, consider the following table:

Region: ALL THE REGIONS				
	BCA	MCA	Others	All the Programmes
2002	9000	8000	45000	62000
2003	9500	7800	43000	60300
2004	6000	9000	42000	57000
2005	4000	9000	40000	53000
ALL the Years	28500	33800	170000	232300

Figure 6: Pivot table on Programme and Year dimension.

The table given above shows the multidimensional data in *cross-tabulation*. This is also referred to as a **pivot table**. Please note that cross-tabulation is a two-dimensional structure. Therefore, if the data warehouse has three dimensions, then the cross-tabulation will be done on only two dimensions and the third dimension would be kept as ALL. For example, the table above has two dimensions Year and Programme, the third dimension Region has a fixed value ALL for the given table. The last row of Figure 6 is on one dimension, viz. Programme.

Now, the question is, how can multidimensional data be represented in a data warehouse? or, more formally, what is the schema for multidimensional data?

Three common multidimensional schemas are the Star schema, the Snowflake schema and Galaxy or Fact Constellation schema. In this Unit, we will define the Star and Snowflake schema. Galaxy schema contains multiple fact tables. You can find more detail on this schema in the further readings. A multidimensional storage model contains two types of tables: the dimension tables and the fact table. The dimension tables have tuples of dimension attributes, whereas the fact tables have one tuple each for a recorded fact. Let us demonstrate this with the help of an example. Consider the University data warehouse where one of the data tables is the *Student_enrolment* table. The three dimensions in such a case would be:

- Year
- Programme, and
- Region

The star schema for such a data is shown in *Figure 7*.

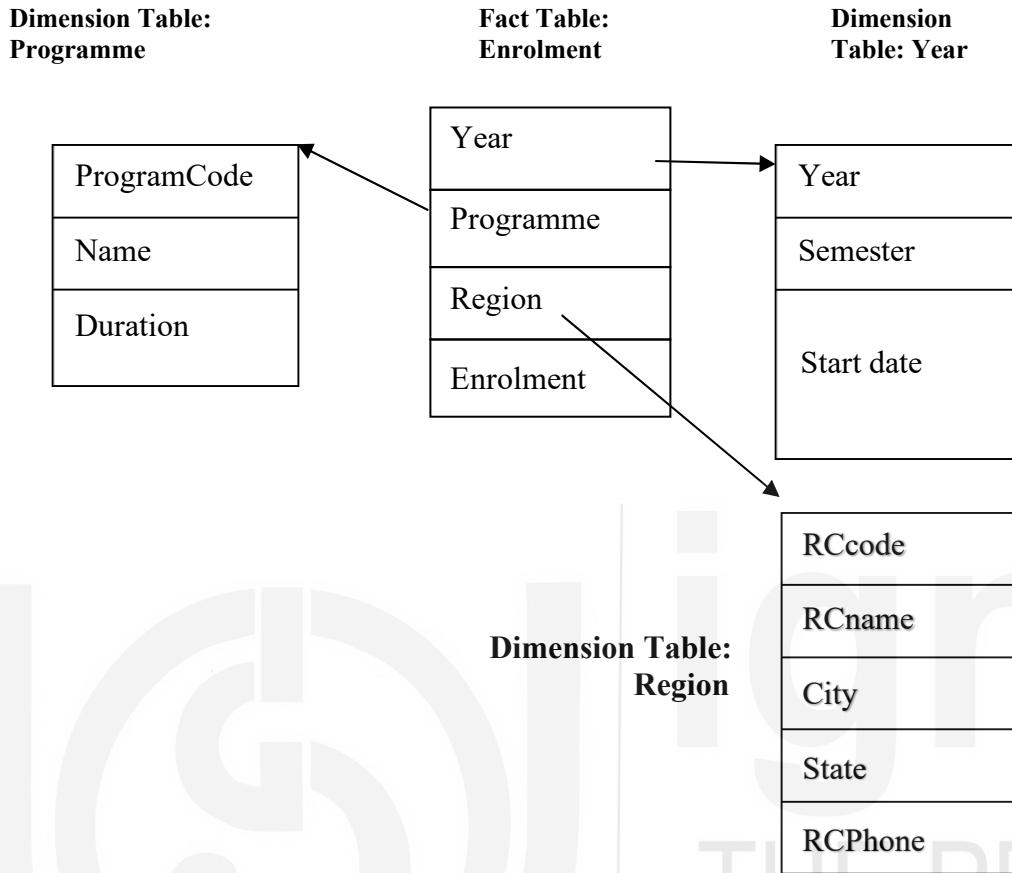


Figure 7: A Star Schema

Please note that in *Figure 7*, the fact table points to different dimension tables, thus, ensuring the reliability of the data. Please also note that each Dimension table is a table for a single dimension only and that is why this schema is known as a Star schema. However, a dimension table may not be normalised. Thus, a new schema named the Snowflake schema was created. A Snowflake schema has normalised but hierarchical dimensional tables. For example, consider the Star schema shown in *Figure 7*, if, in the Region dimension table, the value of the field Rcphone is multivalued, then the Region dimension table is not normalised. Thus, we can create a Snowflake schema for such a situation (Refer to *Figure 8*).

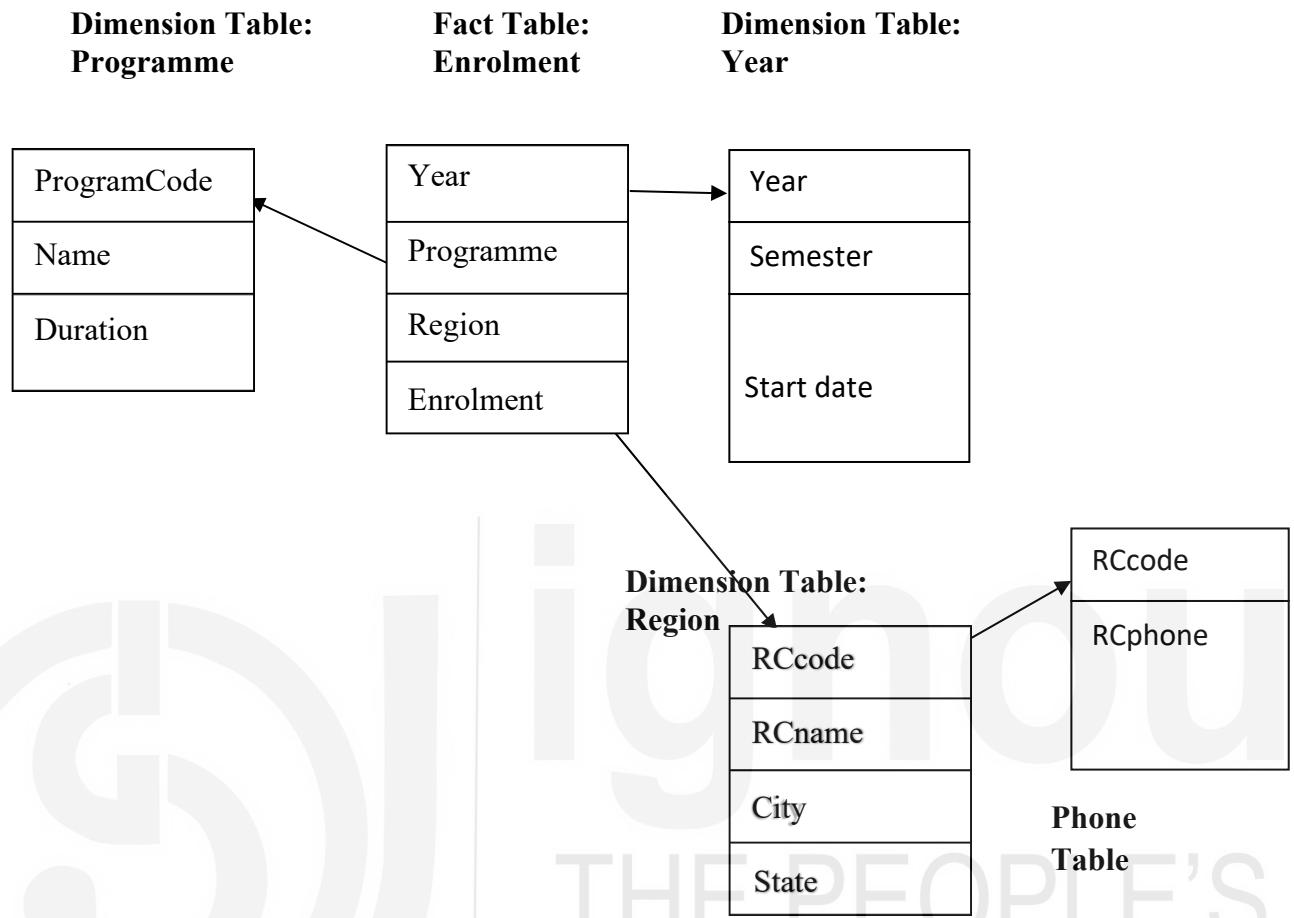


Figure 8: Snowflake Schema

Data warehouse storage can also utilise indexing to support high-performance access. As the data of a data warehouse is non-volatile, the data warehouse allows storage and access to summary data.

A data warehouse is an integrated collection of data and can help the process of making better business decisions. Several tools and methods are available to process the data of a data warehouse to create information and knowledge that supports business decisions. Two such techniques are Decision-support systems and Online Analytical processing. A detailed discussion on these topics is beyond. The scope of this unit. In this unit, we define data mining, which can extract useful information from a data warehouse.

14.5 DATA MINING TECHNOLOGY

Data is growing at a phenomenal rate today and users expect more sophisticated information from data. There is a need for techniques and tools that can automatically generate useful information and knowledge from large volumes of data. Data mining is one such technique of generating hidden information from the large data. Data mining can be defined as: “an automatic process of extraction of non-trivial or implicit or previously unknown but potentially useful information or patterns from data in large databases, data warehouses or in flat files”.

Data mining uses the data of a data warehouse, which is well equipped for providing data as input for data mining. The advantages of using the data of a data warehouse for data mining are listed below:

- Data quality and consistency are essential for data mining to ensure the accuracy of the predictive models. Data is loaded in a data warehouse after data extraction, cleaning and transformation; therefore, is good quality data.
- As stated earlier, the data of a data warehouse is integrated from multiple data sources for a specific purpose and, therefore, is suited for data mining.
- Some data mining techniques would require aggregated or summarised data. A data warehouse contains such data.

As defined earlier, data mining generates potentially useful information or patterns from data. In fact, the information generated through data mining can be used to create knowledge. So let us first define the three terms: data, information and knowledge.

1. **Data (Symbols)** are raw facts, for example, “Miral” is a name, BCA is a programme, and 2200105000 is an enrolment number.
2. **Information:** Information is the processed data. It provides answers to “who”, “what”, “where”, and “when” questions. For example, Miral is a BCA student of IGNOU having enrolment number “2200105000”. He has scored 761 marks out of 1000.
3. **Knowledge:** Knowledge is the application of data and information, it answers the “how” questions. This is not explicit in the database. A student who scores more than 075% marks is brilliant; therefore, Miral is brilliant.

Data Mining Approaches

The approaches to data mining are based on the type of information/ knowledge to be mined. We will emphasise three different approaches: Classification, Clustering, and Association Rules.

The classification task maps data tuples into predefined groups or classes. The task of clustering is to group tuples with similar attribute values into a new class. So, classification is supervised by the goal attribute, while clustering is an unsupervised classification. The task of association rule mining is to find relationships between/among data values of a set of transactional data. Its original application was on “market basket data”.

In most of these approaches, a notion of distance measure is used. A distance measure is used to find the distance or dissimilarity between objects. The two most common distance measures are:

- Euclidean distance:
$$\text{edis}(t_i, t_j) = \sqrt{\sum_{h=1}^k (t_{ih} - t_{jh})^2}$$
- Manhattan distance:
$$\text{mdis}(t_i, t_j) = \sum_{h=1}^k |(t_{ih} - t_{jh})|$$

where t_i and t_j are tuples and h can take the values from 1 to k , each of which represents a different attribute that is being used to compute the distance.

In the next section, we discuss the three data mining approaches.

Check Your Progress – 2

- 1) What is a dimension? How is it different from a fact table?

.....
.....
.....

- 2) How is the Snowflake schema different from the Star schema?

.....
.....
.....

- 3) Define what data mining is.

.....
.....
.....

- 4) What are different data mining tasks?

.....
.....
.....

14.6 CLASSIFICATION

The classification task maps data tuples into predefined groups or classes.

Given a database/dataset consisting of tuples t_i , where i varies from 1 to n , i.e.

$$D = \{t_1, t_2, \dots, t_n\};$$

and a set of known classes $C = \{C_1, \dots, C_m\}$, where $m \gg n$.

The classification problem is to map each t_i to a C_i .

Some simple examples of classification are:

- Teachers classify students' marks data into a set of grades as A, B, C, D, or F.
- You can clarify the height of a set of students in the classes: tall, medium or short.

Basic Principle:

1. The classification involves learning using the training data, which is the data that has already been assigned to one of the classes. This training results in a classification model.
2. This model is then tested using the test data to find the effectiveness of the model. The test data also has assigned classes, which are checked against the class predicted by the model. The accuracy of the model can be ascertained on the basis of correctly predicted classes of the test data.
3. A good model is then used to classify the data that has not been classified.

Some of the common techniques used for classification are Decision Trees, Neural Networks etc. In this section, we present two examples of classification.

14.6.1 Classification Using Distance (K-Nearest Neighbour)

This approach places items in the class to which they are “closest” by determining the distance of an item from a class. Classes are represented by a central point called centroid. The K-nearest neighbour algorithm has the following steps:

- 1) Create a training data set consisting of attributes or features that would be used for classifying data and the identified classes for these attributes. Figure 9 shows an example training data set.
- 2) Defines the number of *near items* (items that have less distance to the attributes of concern) from the training data that should be used to classify data. The value of K should be $\leq \sqrt{\text{Number_of_Training_Items}}$
- 3) A new item is placed in the class in which most of its *near items* are placed.

Example: Consider the following data, which classifies each person’s class <Short, Medium, Tall> depending upon height attribute.

Name	Height	Class
Sunita	1.6m	Short
Ram	2.0m	Tall
Namita	1.9m	Medium
Radha	1.88m	Medium
Jully	1.7m	Short
Arun	1.85m	Medium
Shelly	1.6m	Short
Avinash	1.7m	Short
Sachin	2.2m	Tall
Manoj	2.1m	Tall
Sangeeta	1.8m	Medium
Anirban	1.95m	Medium
Krishna	1.9m	Medium
Kavita	1.8m	Medium
Pooja	1.75m	Medium

Figure 9: Sample Height data with classification

- 1) You are given the task of classifying the tuple xyz <XYZ, 1.6> using the data that is given to you.
- 2) The *height* attribute is used for distance calculation, and suppose K=5, then the following are the five nearest tuples to the tuple xyz. Please note that we have used the Manhattan distance on attribute *height* as a measure of classification.

Height	Class
1.6m	Short
1.7m	Short
1.6m	Short
1.7m	Short
1.75m	Medium

- 3) On examination of the tuples above, we classify the tuple xyz<XYZ, 1.6> to the *Short* class since most of the nearest tuples belong to the *Short* class.

Thus, in K-nearest neighbour classification, the classification is controlled by the neighbours.

14.6.2 Decision Tree

Given a data set $D = \{t_1, t_2, \dots, t_n\}$, where tuple $t_i = \langle A_1, A_2, \dots, A_h \rangle$, that is, each tuple is represented by h attributes. Also, let us suppose that the classes are $C = \{C_1, \dots, C_m\}$, then the Decision or Classification Tree is a tree associated with D such that:

- The tree consists of internal and leaf nodes. A label using an attribute A_i is assigned to each internal node.
- An arc from a parent node is labelled with a predicate on the attribute label of the parent node.
- Every leaf node has a class label.

The basic steps in the Decision Tree are as follows:

- Building the tree by using the training set dataset/database.
- Applying the tree to the new dataset/database.

This decision tree can then be used for classification. We show an example of a decision tree without giving an algorithm for drawing it. You may refer to further reading for details.

Consider the following data in which the *Position* attribute acts as a predicted class, and department and salary are attributes of determining the class of a tuple.

Department	Age	Salary	Position
Personnel	31-40	Medium Range	Boss
Personnel	21-30	Low Range	Assistant
Personnel	31-40	Low Range	Assistant
MIS	21-30	Medium Range	Assistant
MIS	31-40	High Range	Boss
MIS	21-30	Medium Range	Assistant
MIS	41-50	High Range	Boss
Administration	31-40	Medium Range	Boss
Administration	31-40	Medium Range	Assistant
Security	41-50	Medium Range	Boss
Security	21-30	Low Range	Assistant

Figure 10: Sample data for classification

You may analyse the data given in Figure 10 on each individual attribute. The following tables present the analysis of class Position on the attributes Age, Department and Salary, respectively.

Age	Assistant	BOSS
21-30	4	0
31-40	2	3
41-50	0	2

Department	Assistant	BOSS
Personal	2	1
MIS	2	2
Administration	1	1
Security	1	1

Salary	Assistant	BOSS
Low Range	3	0
Medium Range	3	3
High Range	0	2

Out of the three attributes, the Age attribute predicts the Position class the best, as it predicts that all the person in the range 21-30 are Assistants and all the person in the age group 41-50 are Boss. Thus, it should be used as the first splitting attribute. However, for a person in the age range of 31-40, the Postion cannot be defined. So, we

have to find the splitting attribute for this age range 31-40. The tuples that belong to this range are as follows:

Tuples Only for 31-40 age range

Department	Salary	Position
Personnel	Medium Range	Boss
Personnel	Low Range	Assistant
MIS	High Range	Boss
Administration	Medium Range	Boss
Administration	Medium Range	Assistant

The tuples as given above can be mapped as:

Department	Assistant	BOSS
Personal	1	1
MIS	0	1
Administration	1	1

Salary	Assistant	BOSS
Low Range	1	0
Medium Range	1	2
High Range	0	1

The Salary attribute can perform better prediction than the Department attribute, so we select Salary as the next splitting attribute. In the middle range Salary, the Position class is not defined while for other ranges it is defined. So, we have to find the splitting attribute for this middle range. Since only Department attribute is left, so, Department will be the next splitting attribute. Now, the tuples that belong to this salary range are as follows:

Department	Position
Personnel	Boss
Administration	Boss
Administration	Assistant

In the Personnel department, all person are Boss, while, in the Administration there is a tie between the classes. So, the person can be either Boss or Assistant in the Administration department. The final decision tree, based on presented data, for predicting the Position class is as follows:

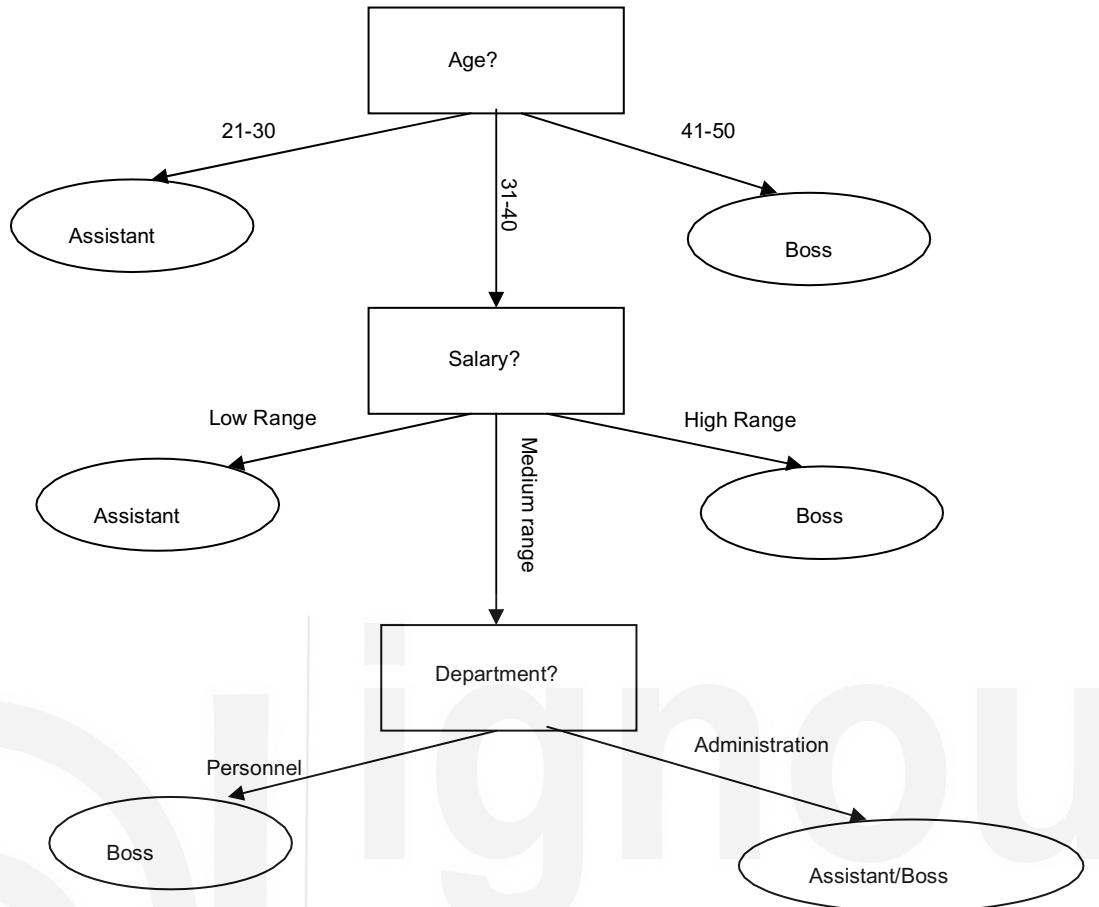


Figure 11: The decision tree for the sample data of Figure 10

Now, you can use the decision tree to predict the Position of a person if his/her Age, Salary and Department attributes are known.

There are many more techniques of classification, however, they are beyond the scope of this unit, you may refer to further readings for these.

14.7 CLUSTERING

Clustering is grouping tuples with similar attribute values into the same group. Given a database $D = \{t_1, t_2, \dots, t_n\}$ of tuples and an integer value K , the Clustering problem is to define a mapping where each tuple t_i is assigned to one cluster K_j , $1 \leq j \leq K$.

Unlike the classification problem, clusters are not known in advance. The user has to enter the value of the number of clusters K . In other words, a *cluster* can be defined as the collection of data objects that are similar in nature, as per certain defining properties, but these objects are dissimilar to the objects in other clusters. Clustering is a very useful exercise, especially for identifying similar groups in the given data. Such data can be about buying patterns, geographical locations, web information and many more. For example, you can use clustering to segment the customer database of a departmental store based on similar buying patterns. Similarly, to identify similar Web usage patterns, you may use clustering. Some of the clustering issues are as follows:

- **Outlier handling:** How will the outliers be handled? (Outliers are data objects that have values far beyond the average data limits of those data objects). Outlier handling requires answering the question: Whether an outlier be

considered or left aside while calculating the clusters?

- **Dynamic data:** How will you handle dynamic data?
 - **Interpreting results:** How will the result be interpreted?
 - **Evaluating results:** How will the result be calculated?
 - **Number of clusters:** How many clusters will you consider for the given data?
 - **Data to be used:** whether you are dealing with quality data or noisy data? If the data is noisy, how is it to be handled?
 - **Scalability:** Whether the algorithm that is used is to be scaled for small as well as large data sets/databases.

There are many algorithms for clustering. However, we will discuss only one basic algorithm. You can refer to more details on clustering from further readings.

Partitioning Clustering

The partitioning clustering algorithm constructs K partitions from a given set of n objects of data. Here, $K \leq n$, and each partition must have at least one data object while one object belongs to **only one** of the partitions. A partitioning clustering algorithm normally requires users to input the desired number of clusters, K . We define one such technique called K -means clustering with the help of an example.

K-Means clustering: In K-Means clustering, initially a set of K clusters is randomly chosen. Then iteratively, items are moved among sets of clusters until the desired set of clusters is reached. A high degree of similarity among elements in a cluster is obtained by using this algorithm. For this algorithm, a set of clusters $K_i = \{t_{i1}, t_{i2}, \dots, t_{im}\}$ is chosen. A cluster mean is computed for each cluster using the equation:

Where t_i represents the tuples and m represents the mean. This mean is used to refine the clusters further. We explain this algorithm with the help of an example.

Example: Consider an input set is given as:

Input set= {1, 2, 3, 5, 10, 15, 16, 18, 22, 30} and $K = 2$.

Find the cluster of input values.

Step 1: Randomly assign means to two clusters, K_1 and K_2 , as $m_1=3$ and $m_2=5$

Assign Input set values to clusters based on the smallest distance to the mean.
This will result in:

$K_1 = \{1, 2, 3\}$ and $K_2 = \{5, 10, 15, 16, 18, 22, 30\}$

Step 2: Compute the mean (use the formula given in equation 1) of the clusters after the assignment of the input set. This will result in $m_1=2$ and $m_2=16.57$.

Redefine clusters as per the closest mean from the mean of the cluster. For example, for the mean of step 2, input value 5 is closer to 2, so add it to K_1 . Thus, the revised clusters are:

$K_1 = \{1, 2, 3, 5\}$ and $K_2 = \{10, 15, 16, 18, 22, 30\}$.

Step 3: Compute the mean for the latest cluster assignment. This will result in $m_1=2.75$ and $m_2=18.5$.

Repeat the process of finding the cluster as per the closest mean from the means of the cluster. The revised clusters remain as:

$K_1 = \{1, 2, 3, 5, 10\}$ and $K_2 = \{15, 16, 18, 22, 30\}$.

Step 4: Compute the mean, it will be $m_1=4.2$, $m_2=20.2$

No change in clusters, so STOP. The final clusters are:

$K_1 = \{1, 2, 3, 5, 10\}$ and $K_2 = \{15, 16, 18, 22, 30\}$.

You can refer to more details on the clustering algorithms from further readings.

☛ Check Your Progress – 3

- 1) What is the classification of data? Give some examples of classification.

.....
.....
.....

- 2) What is clustering?

.....
.....
.....

- 3) How is clustering different from classification?

.....
.....
.....

14.8 ASSOCIATION RULE MINING

The task of association rule mining is to find certain association relationships among a set of items in a dataset/database. The association relationships are described as association rules. In association rule mining there are two measurements, *support* and *confidence*. The confidence measure indicates the rule's strength, while support corresponds to the frequency of the pattern.

A typical example of an association rule created by data mining often termed “market basket data” is “While shopping for basic groceries, it has been found that about 80% of the customers who purchase bread from the store also purchase butter along with.”

Applications of association rule mining include cache customisation, advertisement personalisation, store layout designing, customer segmentation etc. All these applications try to determine the associations between data items, if it exists, to optimise performance.

Formal Definition:

Given a set of items $I = \{i_1, i_2, \dots, i_m\}$ and a set of transactions T , where a transaction T_i comprises of items and is a subset I . Each transaction is identified by a TID. For the given sets, an association rule is defined as $X \rightarrow Y$, where both X and Y are the subsets of I such that $X \cap Y = \emptyset$.

The support (s) for the stated association rule $X \rightarrow Y$ is defined as the percentage of transactions in T that contains $X \cup Y$.

The confidence (c) for the stated association rule $X \rightarrow Y$ is defined as the percentage of transactions that include X , also include Y .

Support indicates how frequently the pattern occurs, while confidence indicates the strength of the rule.

The objective of association rule mining is to find all those rules which have better support than the defined minimum support and better confidence than the defined

minimum confidence. The association rule mining consists of two sub-problems:

- (1) Find the frequent item sets (*FreqItem*) having support more than a predetermined minimum support.
- (2) Derive association rules from *FreqItem*, which have confidence more than the minimum confidence.

There are a lot of ways to find the large item sets, but in this unit, we will only discuss the Apriori Algorithm.

Apriori Algorithm: For finding frequent item sets.

The Apriori algorithm applies the concept that all the subsets of a frequent itemset should be frequent. The Apriori algorithm generates the candidate item sets from the item sets that were found to be frequent in the previous iteration of the algorithm.

This algorithm first finds the frequent item sets having just 1 item, which are combined to form item sets of 2 items and so on. The algorithm iterations terminates when no additional higher item sets can be generated.

Notations that are used in the Apriori algorithm are given below:

k-itemset	An itemset having k items
L_k	Set of frequent k-itemset (those with minimum support)
C_k	Set of candidates k-itemset (for finding frequent item sets)

The Apriori algorithm is implemented as an iterative function. It takes L_{k-1} as an input parameter and returns L_k . It consists of a join step and a pruning step. It is explained with the help of the following example:

Example: Finding frequent item sets:

Consider the following transactions and find the frequent item sets by applying the Apriori algorithm assuming minimum support (s) =30%.

Transaction ID	Item(s) purchased
1	Shirt, Trouser
2	Shirt, Trouser, Coat
3	Coat, Tie, Tiepin
4	Coat, Shirt, Tie, Trouser
5	Trouser, Belt
6	Coat, Tiepin, Trouser
7	Coat, Tie
8	Shirt
9	Shirt, Coat
10	Shirt, Handkerchief

Figure 12: Sample Transactions data

The method of finding the frequent itemset is shown in the Figure 13.

Iteration Number	Candidates	Frequent itemset ($s \geq 3$, i.e. 30% of 10 Transactions)
1	$C_1 = \{ \text{Belt} \ 1, \text{Coat} \ 6, \text{Handkerchief} \ 1, \text{Shirt} \ 6, \text{Tie} \ 3, \text{Tiepin} \ 2, \text{Trouser} \ 5 \}$	$L_1 = \{ \text{Coat} \ 6, \text{Shirt} \ 6, \text{Tie} \ 3, \text{Trouser} \ 5 \}$
2	$C_2 = \{ \{\text{Coat}, \text{Shirt}\} \ 3, \{\text{Coat}, \text{Tie}\} \ 3, \{\text{Coat}, \text{Trouser}\} \ 3, \{\text{Shirt}, \text{Tie}\} \ 1, \{\text{Shirt}, \text{Trouser}\} \ 3, \{\text{Tie}, \text{Trouser}\} \ 1 \}$	$L_2 = \{ \{\text{Coat}, \text{Shirt}\} \ 3, \{\text{Coat}, \text{Tie}\} \ 3, \{\text{Coat}, \text{Trouser}\} \ 3, \{\text{Shirt}, \text{Trouser}\} \ 3 \}$
3	$C_3 = \{ \{\text{Coat}, \text{Shirt}, \text{Trouser}\} \ 2 \}$	$L_3 = \emptyset$

Figure 13: Frequent Itemset for different k using Apriori algorithm

In pass number 1, you may notice that Belt is purchased in only one (5th transaction) of all the 10 transaction that is why it has a value 1. Similarly, the support of each item is computed from the transaction data.

Likewise, in pass 2 {Coat, shirt} together are purchased in transactions 2, 4, 9.

Also please note that set C_2 is computed by joining set L_1 with itself, which will result in

$$C_2: \{ \{\text{Coat}, \text{Shirt}\}, \{\text{Coat}, \text{Tie}\}, \{\text{Coat}, \text{Trouser}\}, \{\text{Shirt}, \text{Tie}\}, \{\text{Shirt}, \text{Trouser}\}, \{\text{Tie}, \text{Trouser}\} \}$$

Set L_2 is computed after finding the support of each element of set C_2 as shown in Figure 13.

The calculation of 3-itemsets is mentioned below:

Join operation on L_2 onto itself yields 3 item sets as:

$$\{ \{\text{Coat}, \text{Shirt}, \text{Tie}\}, \{\text{Coat}, \text{Shirt}, \text{Trouser}\}, \{\text{Coat}, \text{Tie}, \text{Trouser}\} \}$$

However, the Prune operation removes two of these items from the candidate set C_3 due to the following reasons:

- $\{\text{Coat}, \text{Shirt}, \text{Tie}\}$ is pruned as one of its subset $\{\text{Shirt}, \text{Tie}\}$ is not in L_2
- $\{\text{Coat}, \text{Shirt}, \text{Trouser}\}$ is retained as $\{\text{Coat}, \text{Shirt}\}$, $\{\text{Coat}, \text{Trouser}\}$ and $\{\text{Shirt}, \text{Trouser}\}$ all three are in L_2
- $\{\text{Coat}, \text{Tie}, \text{Trouser}\}$ is pruned as its subset $\{\text{Tie}, \text{Trouser}\}$ is not in L_2

The algorithm terminates, as L_3 is a NULL set. Thus, the frequent 2 items are:

$$L_2 = \{ \{\text{Coat}, \text{Shirt}\}, \{\text{Coat}, \text{Tie}\}, \{\text{Coat}, \text{Trouser}\}, \{\text{Shirt}, \text{Trouser}\} \}$$

Finding Association Rules: Assuming minimum confidence (c) of 60%.

Confidence for rule $\text{Coat} \rightarrow \text{Shirt}$

$$= \frac{\text{Transactions containing } \{\text{Coat}, \text{Shirt}\}}{\text{Transactions containing only } \{\text{Coat}\}} \times 100 \\ = \frac{3}{6} \times 100 = 50\%$$

$$\text{Confidence for rule } \text{Shirt} \rightarrow \text{Coat} = \frac{3}{6} \times 100 = 50\%$$

$$\text{Confidence for rule } \text{Coat} \rightarrow \text{Tie} = \frac{3}{6} \times 100 = 50\%$$

$$\text{Confidence for rule } \text{Tie} \rightarrow \text{Coat} = \frac{3}{3} \times 100 = 100\%$$

$$\text{Confidence for rule Coat} \rightarrow \text{Trouser} = \frac{3}{6} \times 100 = 50\%$$

$$\text{Confidence for rule Trouser} \rightarrow \text{Coat} = \frac{3}{5} \times 100 = 60\%$$

$$\text{Confidence for rule Shirt} \rightarrow \text{Trouser} = \frac{3}{6} \times 100 = 50\%$$

$$\text{Confidence for rule Trouser} \rightarrow \text{Shirt} = \frac{3}{5} \times 100 = 60\%$$

Only the following three association rules fulfil the confidence criteria of confidence $\geq 60\%$.

Tie \rightarrow Coat; Trouser \rightarrow Coat; Trouser \rightarrow Shirt

Thus, the Apriori algorithm is able to determine the frequent item sets, which can be used to determine the association rules. One of the major advantages of the Apriori algorithm is that it is a very easy algorithm to implement; however, it requires the transaction database to be memory resident.

14.9 APPLICATIONS OF DATA MINING

Some of the applications of data mining are as follows:

- **Marketing and sales data analysis:** A company can use customer transactions in their database to segment the customers into various types. Such companies may launch products for specific customer types.
- **Investment analysis:** Customers can look at the areas where they can get good returns by applying data mining.
- **Loan approval:** Companies can generate rules for giving loans depending upon the dataset they have. On that basis, they may decide to whom and what amount of loan should be given.
- **Fraud detection:** By finding the correlation between faults, new faults can be detected by applying data mining.
- **Network management:** By analysing patterns generated by data mining for the networks and their faults, the faults can be minimised as well as future hardware and software needs of the network can be predicted.
- **Brand Loyalty:** Given a customer and the product he/she uses, predict whether the customer will change their products.

☛ Check Your Progress – 4

- 1) What is association rule mining?

.....
.....
.....

- 2) What are the applications of data mining in the banking domain?

.....
.....
.....

- 3) Apply the Apriori algorithm for generating frequent itemset in the following dataset:

Transaction ID	Items purchased
T _A	P _I ₂ P _I ₅
T _B	P _I ₁ P _I ₃
T _C	P _I ₁ P _I ₂ P _I ₃ P _I ₄
T _D	P _I ₁ P _I ₂ P _I ₃

14.10 SUMMARY

This unit first introduces the concepts of data warehousing systems. The data warehouse is a technology that collects operational data from several operational systems, refines it and stores it in its own multidimensional model, such as Star schema or Snowflake schema. The data of a data warehouse can be indexed and can be used for analyses through data mining. Data mining is the process of automatic extraction of interesting but not known information in large databases. Basic data mining tasks are Classification, Clustering and Association rules. The classification task maps data into predefined classes. Clustering task groups objects with similar properties/behaviour into the same group. Association rules find the association relationship among a set of objects. Data mining can be applied in many areas, whether it is Games, Marketing, Bioscience, Loan approval, Fraud detection etc.

Please go through further readings for more details on data warehousing and data mining.

14.11 SOLUTIONS/ANSWERS

☛ Check Your Progress – 1

- 1) A Data Warehouse is a repository of processed but integrated information that can be used for queries and analysis. Data and information are extracted from heterogeneous sources. It is subject-oriented, time-variant integrated data, which is not changed once put in a data warehouse.
- 2)
 - Integrated data
 - Subject-oriented data
 - Time-variant data
 - Non-volatile
- 3) ETL is Extraction, transformation, and loading. ETL refers to the methods involved in accessing and manipulating data available in various sources and loading it into the target data warehouse. The following are some of the transformations that may be used during ETL:
 - Filter Transformation
 - Joiner Transformation
 - Aggregator transformation
 - Sorting transformation.

Check Your Progress – 2

- 1) A dimension may be equated with a reference object. For example, in a sales organisation, the dimensions may be *salesperson*, *product* and *period* of information. Each of these is a dimension. The fact table will represent the fact relating to the dimensions. For this example, a fact table will include *sales* (in rupees) made by a particular *salesperson* for a specific *product* for a certain *period*. Fact is actual data. A fact, thus, represents an aggregation of relational data on the dimensions.
- 2) The primary difference between them is that the Snowflake schema uses a normalised dimensional table.
- 3) Data mining is the process of automatic extraction of interesting (non-trivial, implicit, previously unknown and potentially useful) information or patterns from large data stored in large databases or data warehouses.
- 4) Different data-mining tasks are - Classification, Clustering and Association Rule Mining.

Check Your Progress – 3

- 1) The classification task maps data into predefined groups or classes. Data are represented as tuples, which consist of a set of predicated attributes and a goal attribute. The task is to discover some kind of relationship between the predicated attributes and the goal attribute so that the discovered knowledge can be used to predict the class of new tuple(s).

Some examples of classification are:

- Classification of students' grades depending on their marks in previous examinations.
- Classification of customers as good or bad customers in a bank.

- 2) The task of clustering is to group the tuples with similar attribute values into the same class. Given a database of tuples and an integer value K , Clustering defines mapping, such that tuples are mapped to K different clusters.
- 3) In classification, the classes are predetermined, but in the case of clustering, the groups are not predetermined. Only the number of clusters is decided by the user.

Check Your Progress – 4

- 1) The basic idea of association rule mining is to find unknown and interesting associations among various data items.
- 2) Data mining applications in banking are as follows:
 - 1) Detecting patterns of fraudulent credit card use.
 - 2) Identifying good customers.
 - 3) Determining whether to issue a credit card to a person or not.
 - 4) Finding hidden correlations between different financial indicators.
- 3) The dataset D given for the problem is:

Transaction ID	Items purchased
T _A	P _I ₂ P _I ₅
T _B	P _I ₁ P _I ₃
T _C	P _I ₁ P _I ₂ P _I ₃ P _I ₄
T _D	P _I ₁ P _I ₂ P _I ₃

Assuming the minimum support as 50% for calculating the large item sets. As we have 4 transactions, at least 2 transactions should have the data item.

First Scan:

C₁: P_I₁:3, P_I₂:3, P_I₃:3, P_I₄:1, P_I₅:1
 L₁: P_I₁:3, P_I₂:3, P_I₃:3
 C₂: P_I₁P_I₂, P_I₁P_I₃, P_I₂P_I₃

Second Scan:

C₂: P_I₁P_I₂:2, P_I₁P_I₃:3, P_I₂P_I₃:2
 L₂: P_I₁P_I₂:2, P_I₁P_I₃:3, P_I₂P_I₃:2
 C₃: P_I₁P_I₂P_I₃
 Pruned C₃: P_I₁P_I₂P_I₃

Third scan

L₃: P_I₁P_I₂P_I₃: 2

Frequent item sets L={L₁, L₂, L₃}

14.12 FURTHER READINGS

-
- 1) *Data Mining Concepts and Techniques*, J Han, M Kamber, Morgan Kaufmann Publishers, 2001.
 - 2) *Data Mining*, A K Pujari, 2004.

UNIT 15 NoSQL DATABASE

Structure	Page Nos.
15.0 Introduction	
15.1 Objectives	
15.2 Introduction to NoSQL	
15.2.1 What is NoSQL	
15.2.2 Brief History of NoSQL Databases	
15.2.3 NoSQL Database Features	
15.2.4 Difference between RDBMS and NoSQL	
15.3 Types of NoSQL Databases	
15.3.1 Column Based	
15.3.2 Graph Based	
15.3.3 Key-Value Pair Based	
15.3.4 Document Based	
15.4 Summary	
15.5 Solutions/Answers	
15.6 Further Readings	

15.0 INTRODUCTION

In the previous Unit of this Block, you have gone through the concept of relational database management systems and data warehousing. However, these technologies are somewhat slower for scalable web applications. NoSQL databases arose because databases at the time were not able to support the rapid development of scalable web-based applications.

NoSQL databases have changed the manner in which data is stored and used, despite the fact that relational databases are still commonly employed. Most applications come with features like Google-style search, for instance. The growth of data, online surfing, mobile use, and analytics have drastically altered the requirements of contemporary databases. These additional requirements have spurred the expansion of NoSQL databases, which now include a range of types such as key-value, document, column, and graph.

In this Unit, we will discuss the many kinds of NoSQL databases, including those that are built on columns, graphs, key-value pairs, and documents respectively.

15.1 OBJECTIVES

After going through this unit, you should be able to:

- define what is NoSQL;
- differentiate between NoSQL and SQL;
- explain the basic features of Column based NoSQL Database;
- explain the Graph-based NoSQL Database;
- explain the Key-value pair based NoSQL Database and
- explain the Document based NoSQL Database.

15.2 INTRODUCTION TO NoSQL

Databases are a crucial part of many technological and practical systems. The phrase "NoSQL database" is frequently used to describe any non-relational database. NoSQL is sometimes referred to as "non SQL," but it is also referred to as "not only SQL." In either case, the majority of people agree that a NoSQL database is a type of database that stores data in a format that is different from relational tables.

Whenever you want to use the data, it must first be saved in a particular structure and then converted into a usable format. On the other hand, there are some circumstances in which the data are not always presented in a structured style, which means that their schemas are not always rigorous. This unit provides an in-depth look into NoSQL and the features that make it unique.

15.2.1 What is NoSQL?

NoSQL is a way to build databases that can accommodate many different kinds of information, such as key-value pairs, multimedia files, documents, columnar data, graphs, external files, and more. In order to facilitate the development of cutting-edge applications, NoSQL was designed to work with a variety of different data models and schemas.

The great functionality, ease of development, and performance at scale offered by NoSQL have helped make it a popular name. NoSQL is sometimes referred to as a non-relational database due to the numerous data handling features it offers. Due to the fact that it does not adhere to the guidelines established by Relational Database Management Systems (RDBMS), you cannot query your data using conventional SQL commands. We can think of such well-known examples as MongoDB, Neo4J, HyperGraphDB, etc.

15.2.2 Brief History of NoSQL Databases

In the late 2000s, as the price of storage began to plummet, No-SQL databases began to gain popularity. No longer is it necessary to develop a sophisticated, difficult-to-manage data model to prevent data duplication. Because developers' time was quickly surpassing the cost of data storage, NoSQL databases were designed with efficiency in mind.

Table 1: History of Databases

Year	Database Solutions	Company / Database Technology
1970-2000	Mainly RDBMS related	Oracle, IBM DB2, SQL Server, MySQL
2000-2005	DotCom boom – new scale solutions, start of NoSQL dev, whitepapers	Google, Facebook, IBM, amazon
2005-2010	New open source & mainstream databases	Cassandra, Riak, Apache Hbase, neo4j, MongoDB, CouchDB, Redis

2010 onwards	Adoption of Cloud	DBaaS (Database as a Service)
--------------	-------------------	-------------------------------

As storage costs reduced significantly, the quantity of data that applications were required to store and query grew. This data came in all forms—structured, semi-structured, and unstructured—and sizes making it practically difficult to define the schema in advance. NoSQL databases give programmers a great deal of freedom by enabling them to store enormous amounts of unstructured data.

In addition, the Agile Manifesto was gaining momentum, and software developers were reconsidering their approach to software development. They were beginning to understand the need of being able to quickly adjust to ever-evolving requirements. They required the flexibility to make rapid iterations and adjustments to all parts of their software stack, including the underlying database. They were able to achieve this flexibility because of NoSQL databases.

The use of the public cloud as a platform for storing and serving up data and applications was another trend that arose, as cloud computing gained popularity. To make their applications more robust, to expand out rather than up, and to strategically position their data across geographies, they needed the option to store data across various servers and locations. These features are offered by some NoSQL databases like MongoDB.

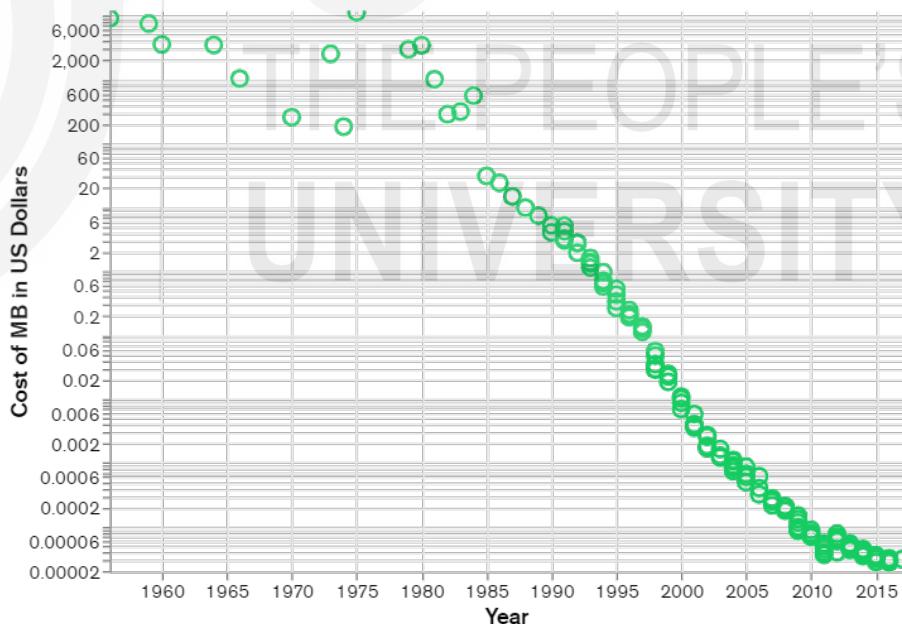


Figure 1: Cost per MB of Data over Time (Log Scale)
(Adapted from <https://www.mongodb.com>)

15.2.3 NoSQL database features

Every NoSQL database comes with its own set of one-of-a-kind capabilities. The following are general characteristics shared by several NoSQL databases:

- Schema flexibility

- Horizontal scaling
- Quick responses to queries as a result of the data model
- Ease of use for software developers

15.2.4 Difference between RDBMS and NoSQL

The differences and similarities between the two DBMSs are as follows:

- For the most part, NoSQL databases fall under the category of non-relational or distributed databases, while SQL databases are classified as Relational Database Management Systems (RDBMS).
- Databases that use the Structured Query Language (SQL) are table-oriented, while NoSQL databases use either document-oriented or key-value pairs or wide-column stores, or graph databases.
- Unlike NoSQL databases, which have dynamic or flexible schema to manage unstructured data, SQL databases have a strict or static schema.
- Structured data is stored using SQL, whereas both structured and unstructured data can be stored using NoSQL.
- SQL databases are thought to be scalable in a vertical direction, whereas NoSQL databases are thought to be scalable in a horizontal direction.
- Increasing the computing capability of your hardware is the first step in the scaling process for SQL databases. In contrast, NoSQL databases scale by distributing the load over multiple servers.
- MySQL, Oracle, PostgreSQL, and Microsoft SQL Server are all examples of SQL databases. BigTable, MongoDB, Redis, Cassandra, RavenDb, Hbase, CouchDB, and Neo4j are a few examples of NoSQL databases.

Vertical scalability is required for SQL databases. This means that an excessive amount of load must be able to be managed by increasing the amount of CPU, SSD, RAM, GPU, etc. on your server. When it comes to NoSQL databases, the ability to scale horizontally is one of their defining characteristics. This means that the addition of additional servers will make the task of managing demand more manageable.

Check Your Progress 1

1) What is NoSQL?

.....

2) What are the features of NoSQL databases?

.....

- 3) Differentiate between the NoSQL and SQL.
-
.....
.....

15.3 TYPES OF NoSQL DATABASES

In this section, we will discuss the many classifications of NoSQL databases. There are typically four types of NoSQL databases:

- 1) Column-based: Instead of accumulating data in rows, this method organizes it all together into columns, which makes it easier to query large datasets.
- 2) Graph-based: These are systems that are utilized for the storage of information regarding networks, such as social relationships.
- 3) Key-value pair based: This is the simplest sort of database, in which each item of your database is saved in the form of an attribute name (also known as a "key") coupled with the value.
- 4) Document-based: Made up of sets of key-value pairs that are kept in documents.

15.3.1 Column Based

A column store, in contrast to a relational database, is arranged as a set of columns, rather than rows. This allows you to read only the columns you need for analysis, saving memory space that would otherwise be taken up by irrelevant information. Because columns are frequently of the same kind, they are able to take advantage of more efficient compression, which makes data reading even quicker. The value of a specific column can be quickly aggregated using columnar databases.

Although columnar databases are excellent for analytics, because of the way they publish data, it is challenging for them to remain consistent because writes to all the columns need several write events on the disk. However, this problem never arises with relational databases because row data is continuously written to disk.

How Does a Column Database Work?

A columnar database is a type of database management system (DBMS) that allows data to be stored in columns rather than rows. It is accountable for reducing the amount of time needed to return a certain query. Additionally, it is accountable for the significant enhancement of the disk I/O performance. Both data analytics and data warehousing benefit from it. Additionally, the primary goal of a Columnar Database is to read and write data in an efficient manner. Column-store databases include Casandra, CosmoDB, Bigtable, and HBase, to name a few.

Columnar Database Vs Row Database:

When processing big data analytics and data warehousing, there are a number of different techniques that can be used, including columnar databases and row databases. But they each take a different method.

For instance:

- Row Database: “Customer 1: Name, Address, Location”. (The fields for each new record are stored in a long row).
- Columnar Database: “Customer 1: Name, Address, Location”. (Each field has its own set of columns). Refer Table 2 for relational database example.

Table 2: Relational database: an example

ID Number	First Name	Last Name	Amount
A01234	Sima	Kaur	4000
B03249	Tapan	Rao	5000
C02345	Srikant	Peter	1000

In a Columnar DBMS, the data will be stored in the following format:

A01234, B03249, C02345; Sima, Tapan, Srikant; Kaur, Rao, Peter; 4000, 5000, 1000.

In a Row-oriented DBMS, the data will be stored in the following format:

A01234, Sima, Kaur, 4000; B03249, Tapan, Rao, 5000; C02345, Srikant, Peter, 1000.

Columnar databases: advantages

The use of columnar databases has various advantages:

- Column stores are highly effective in compression, making them storage efficient. This implies that you can conserve disk space while storing enormous amounts of data in a single column.
- Aggregation queries are fairly quick with column-store databases because the majority of the data is kept in a column, which is beneficial for projects that need to execute a lot of queries quickly.
- Load times are also quite good; a table with a billion rows can be loaded in a matter of seconds. This suggests that you can load and query practically instantly.
- A great deal of versatility because columns do not have to resemble one another. The database would not be affected if you add new or different columns, however, updating all tables is necessary to input whole new record queries.
- Overall, column-store databases are excellent for analytics and reporting due to their quick query response times and capacity to store massive volumes of data without incurring significant costs.

Column databases: Disadvantages

While there are many benefits to adopting column-oriented databases, there are also a few drawbacks to keep in mind.

- It takes a lot of time and effort to create an efficient indexing schema.
- Incremental data loading is undesirable and is to be avoided, if at all possible, even though this might not be a problem for some users.
- This applies to all forms of NoSQL databases, not just those with columns. Web applications frequently have security flaws, and the absence of security features in NoSQL databases does not help. If security is your top goal, you should either consider using relational databases or, if it's possible, use a clearly specified schema.
- Due to the way data is stored, Online Transaction Processing (OLTP) applications are incompatible with columnar databases.

Are columns databases always NoSQL?

Before we conclude, we should note that column-store databases are not always NoSQL-only. It is frequently argued that column-store belongs firmly in the NoSQL camp because it differs so much from relational database approaches. The debate between NoSQL and SQL is generally quite nuanced, therefore this is not usually the case. They are essentially the same as SQL techniques when it comes to column-store databases. For instance, `keyspaces` function as schema, so schema management is still necessary. A NoSQL data store's `keyspace` contains all column families. The concept is comparable to relational database management systems' schema. There is typically only one `keyspace` per program. Another illustration is the fact that the metadata occasionally resembles a conventional relational DBMS perfectly. Ironically, column-store databases frequently adhere to ACID and SQL standards. However, NoSQL databases are often either document-store or key-store, neither of which are column-store. Therefore, it is difficult to claim that column-store is a pure NoSQL system.

15.3.2 Graph Based

The initial hardware hurdles that made it feasible for SQL to handle vast quantities of data are no longer there, despite the fact that SQL is an excellent superb RDBMS and has been used for many years to manage massive amounts of data. As a result, NoSQL has rapidly emerged as the dominant form of contemporary database management and many of the largest websites, we rely on today, are powered by NoSQL, like Twitter's use of FlockDB and Amazon's DynamoDB.

A database that stores data using graph structures is known as a graph database. It represents and stores data using nodes, edges, and attributes rather than tables or documents. Relationships between the nodes are represented by the edges. This makes data retrieval simpler and, in many circumstances, only requires one action. Additionally, it works fantastically as a database for fast, threaded data structures like those used on Twitter

How does a Graph Database Work?

Graphs, which are not relational databases, rely heavily on the idea of multi-

relational data "pathways" for their functionality. However, the structure of graph databases is typically simple. They are largely made up of two elements:

- The Node: This represents the actual data itself. It may be the number of people who watched a video on YouTube; it could be the number of people who read a tweet; or it could even be fundamental information like people's names, addresses, and other such details.
- The Edge: This clarifies the real connection between the two nodes. It is interesting to note that edges can also have their own data, such as the type of connection between two nodes. Similar to edges, mentioned directions may also describe the direction in which the data is flowing.

Graph databases are mostly utilized for studying relationships. For instance, businesses might extract client information from social media using a graph database. For example, some organization might use a graph database to extract data about relationships between Person, Restaurant, and City, as shown in Figure 2.

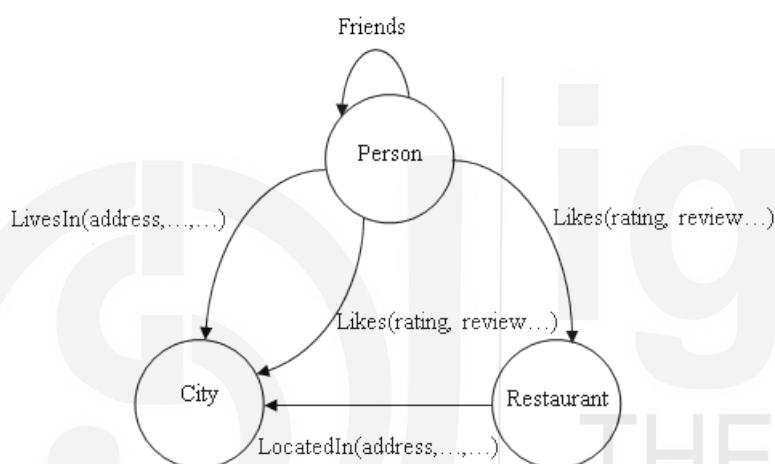


Figure 2. Different Nodes and Edges in Graph Database.

(Adapted from <https://www.kdnuggets.com/>)

When do we need Graph Database?

- 1) It resolves issues with many-to-many relationships. For example, many-to-many relationships include friends of friends.
- 2) When connections among data pieces are more significant. For example, there is a profile with some unique information, but the main selling point is the relationship between these different profiles, which is how you get connected inside a network.
- 3) Low latency with big amounts of data. The relational database's data sets will grow significantly as you add more relationships, and when you query it, its complexity will increase and it will take longer than usual. However, graph databases are specifically created for this purpose, and one can easily query relationships.

Now, let's look at a more specific illustration to explain a group of people's complicated relationships. For example, five friends share a social network. These friends are Binny, Bhawna, Chaitanya, Manish, and Mohit. Their personal data may be kept in a graph database that resembles this, as shown in Figure 3 and Table 3:

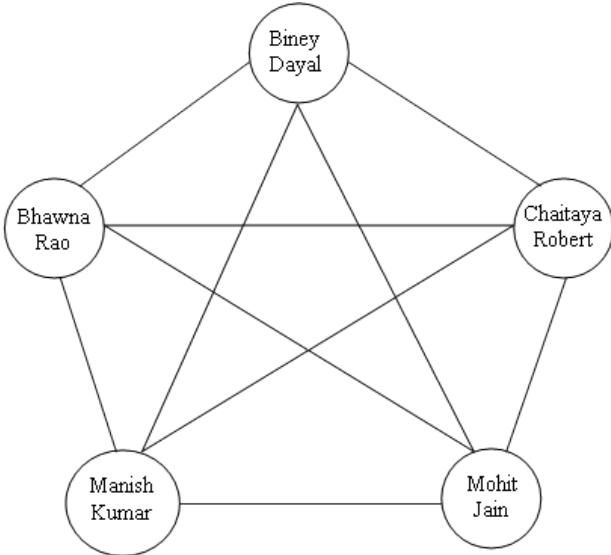


Figure 3. Example-Five friends sharing Social network.

Table 3: Relational database: an example

Id	Firstname	Lastname	Email	Mobile
1001	Biney	Dayal	binnya@example.com	8645212321
1002	Bhawna	Rao	bhawanrao@example.com	9645212323
1003	Chaitaya	Robert	chaitayarob@example.com	7645212356
1004	Manish	Kumar	mkumar@example.com	9955212320
1005	Mohit	Jain	mjain@example.com	9945212329

This means we will need yet another table to keep track of user relationships. Our friendship table (refer Table 4) will resemble the following:

Table 4: Friendship Table

user id	friend id
1001	1002
1001	1003
1001	1004
1001	1005
1002	1001
1002	1003
1002	1004
1002	1005
1003	1001
1003	1002
1003	1004
1003	1005
1004	1001
1004	1002
1004	1003
1004	1005
1005	1001
1005	1002
1005	1003
1005	1004

We won't go too deeply into the theory of the database's main key and foreign key. Instead, presume that the friendship table uses both friends' ids. Let's say that every member on our social network gets access to a feature that lets them view the personal information of their other users who are friends with them. This means that if Chaitaya were to ask for information, it would be regarding Biney, Bhawna, Manish and Mohit. We shall address this issue in a conventional

(relational database) manner. First, we need to locate Chaitaya's user id in the database's Users table (refer Table 5).

Table 5: Chaitaya's Record

Id	Firstname	Lastname	Email	Mobile
1003	Chaitaya	Robert	chaitanyarob@example.net	7645212356

We would now search the friendship table (refer Table 6) for all tuples with the user id of 3. The resulting relationship would look like this:

Table 6: Friendship Table for user id 3

user_id	friend_id
1003	1001
1003	1002
1003	1004
1003	1005

Let us now examine the time required for this Relational database strategy. This will be close to $\log(N)$ times, where N is the number of tuples in the friendship table. In this case, the database continues to keep the entries in sequential order based on their ids. So, in general, the time complexity for 'M' number of queries is $M * \log(N)$. Only, if we had used a graph database strategy the overall time complexity has been $O(N)$. For the simple reason that once Chaitaya has been located in the database, all the rest of her friends may be found with a single click, as shown in Figure 4.

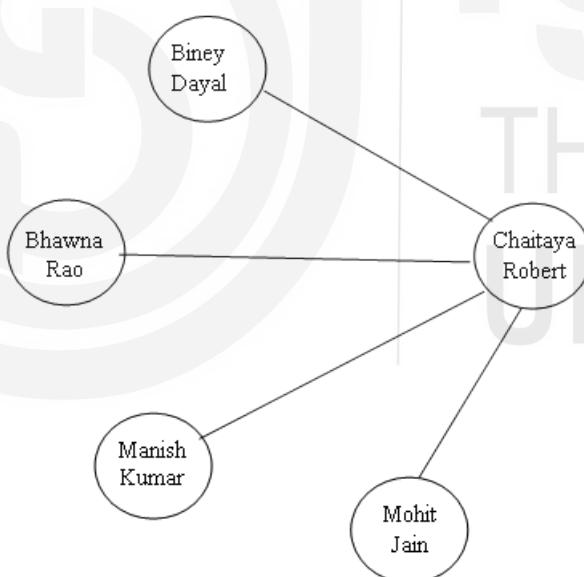


Figure 4. Accessing other data with a single click.

Graph Database Examples

Although graph databases are not as widely used as other NoSQL databases, there are a handful that have become de facto standards when discussing NoSQL:

Neo4j is both an open-source and an interestingly developed on Java graph database. It is considered to be one of the best graph databases. In addition to that, it comes with its own language known as Cypher, which is comparable to the declarative SQL language but is designed to work with graphs. In addition to Java, it supports a number of other popular programming languages, including Python, .NET, JavaScript, and a few others. Neo4j excels in applications such as

the administration of data centers and the identification of fraudulent activity.

RedisGraph is a graph module that is integrated into Redis, which is a key-value NoSQL database. RedisGraph was developed to have its data saved in RAM for the same reason that Redis itself is constructed on in-memory data structures. As a result, a graph database with excellent speed and quick searching and indexing is created. RedisGraph also makes use of Cypher, which is ideal if you're a programmer or data scientist looking for greater database flexibility. Applications that require blazing-fast performance are the main uses.

OrientDB It is interesting to note that OrientDB supports graph, document store, key-value store, and object-based data formats. Having stated that, the graph model, which uses direct links between databases, is used to hold all of the relationships. Although it does not use Cypher, OrientDB is open-source and developed in Java, just like Neo4j and the two prior graph databases. OrientDB is designed to be used in situations when many data models are necessary, and as a result, it is optimized for data consistency as well as minimizing data complexity.

15.3.3 Key-value pair Based

Key-value stores are perhaps the most widely used of the four major NoSQL database formats because of their simplicity and quick performance. Let us examine key-value stores' operation and application in more detail. With some of the most well-known platforms and services depending on them to deliver material to users with lightning speed, NoSQL has grown in significance in our daily lives. Of course, NoSQL includes a range of database types, but key-value store is unquestionably the most used.

Because of its extreme simplicity, this kind of data model is built to execute incredibly quickly when compared to relational databases. Furthermore, because key-value stores adhere to the scalable NoSQL design philosophy, they are flexible and simple to set up.

How Does a Key-Value Work?

In reality, key-value storage is quite simple. A value is saved with a key that specifies its location, and a value can be pretty much any piece of data or information. In reality, this design idea may be found in almost every programming language as an array or map object, refer Figure 5. The fact that it is persistently kept in a database management system makes a difference in this case.

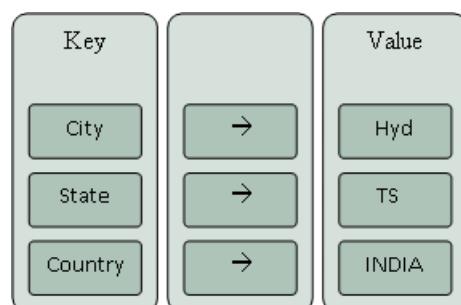


Figure 5. Example Key-Value database.

Popularity of key-value stores is due to the fact that information is stored as a single large piece of data instead of as discrete data. As a result, indexing the database is not really necessary to improve its performance. Instead, because of

the way it is set up, it operates more quickly on its own. Similar to that, it mostly uses the get, put, and delete commands rather than having a language of its own. Of course, this has the drawback that the data you receive in response to a request is not screened. Under certain conditions, this lack of data management may be problematic, but generally speaking, the trade-off is worthwhile. Because key-value stores are both quick and reliable, the vast majority of programmers find ways to get around any filtering or control problems that may arise.

Benefits of Key-Value

Key-value data models, one of the more well-liked types of NoSQL data models, provide many advantages when it comes to creating a database:

Scalability: Key-value stores, like NoSQL in general, are infinitely scalable in a horizontal fashion, which is one of its main advantages over relational databases. This can be a huge advantage for sophisticated and larger databases compared to relational databases, where expansion is vertical and finite, as shown in Figure 6.

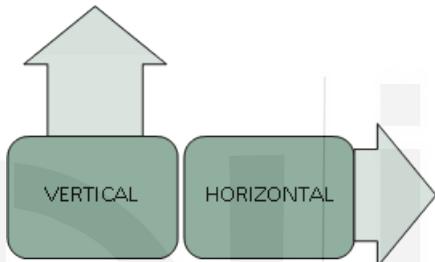


Figure 6. Horizontal and Vertical Scalability.

More specifically, partitioning and replication are used to manage this. Additionally, by avoiding things like low-overhead server calls, it decreases the ACID guarantees.

No/Simpler Querying: With key-value stores, querying is really not possible except in very particular circumstances when it comes to querying keys, and even then, it is not always practicable. Because there is just one request to read and one request to write, key-value makes it easier to manage situations like sessions, user profiles, shopping carts, and so on (due to the blob-like nature of how the data is stored). Similar to this, concurrency problems are simpler to manage because only one key needs to be resolved.

Mobility: Because key-value stores lack a query language, it is simple to move them from one system to another without modifying the architecture or the code. Thus, switching operating systems is less disruptive than switching relational databases.

When to Use Key-Value

Key-value stores excel in this area because traditional relational databases are not actually designed to manage a large number of read/write operations. Key-value can readily scale to thousands of users per second due to its scalability. Additionally, it can easily withstand lost storage or data because of the built-in redundancy.

As a result, key-value excels in the following instances:

- Profiles and user preferences
- Large-scale user session management
- Product suggestions (such as in eCommerce platforms)

- Delivery of personalized ads to users based on their data profiles
- Cache data for infrequently updated data

There are numerous other circumstances where key-value works nicely. For instance, because of its scalability, it frequently finds usage in big data research. Similar to how it works for web applications, key-value is effective for organizing player sessions in MMOG (massively multiplayer online game) and other online games.

Key-Value Database Examples

Some key-value database models, for instance, save information to a solid-state drive (SSD), while others use random-access memory (RAM). We depend on key-value stores on a daily basis in our lives since they are some of the most popular and frequently used databases. The fact is that some of the most popular and commonly used databases are key-value stores.

Amazon DynamoDB is most likely the database that is used the most often for key-value storage. In point of fact, study into Amazon DynamoDB was the impetus for the rise in popularity of NoSQL.

Aerospike is a free and open-source database that was designed specifically for use with in-memory data storage.

Berkeley DB: Another free and open-source database, Berkeley DB is a high-performance framework for storing databases, despite the fact that it has a very simple interface.

Couchbase: Text searches and querying in a SQL-like format are both possible with Couchbase, which is an interesting feature.

Memcached not only saves cached data in RAM, which helps websites load more quickly, but it is also free and open source.

Riak was designed specifically for use in the app development process, and it plays well with other databases and app platforms.

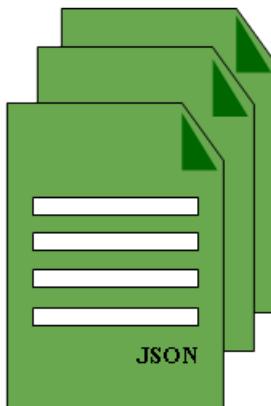
Redis: A database that serves as both a memory cache and a message broker.

15.3.4 Document Based

A non-relational database that stores data as structured documents is known as a document database (also known as a NoSQL document store). Instead of using standard rows and columns, JSON format is a more recent technique to store data. An XML or JSON file, or a PDF, are all examples of documents. NoSQL is everywhere nowadays; just look at Twitter and its use of FlockDB or Amazon and their use of DynamoDB. Figure 7 shows the difference between the Relational and Document Store model.

C1	C2	C3

Relational Data Model



Document Store Model

Figure 7: Relational Vs Document Store Model.

In spite of the fact that there are a great deal of data models, each of which contains hundreds of databases, the one we are going to investigate today is called Document-store. One of the most common database models now in use, document-store functions in a manner that is somewhat similar to that of the key-value model in the sense that documents are saved together with particular keys that access the information. Figure 8 (a) shows the document that holds information about a book. This file is a JSON representation of a book's metadata, which includes the book's BookID, Title, Author, and Year and Figure 8 (b) shows the same metadata for Key value database.

A Document
{ "BookID": "978-1449396091", "Title": "DBMS", "Author": "Raghu Ramakrishnan", "Year": "2022", }

(a)

Key	Value
BookID	978-1449396091
Title	DBMS
Author	Raghu Ramakrishnan
Year	2022

(b)

Figure 8: Example of Document and Key-value database

When to use a document database?

- When your application requires data that is not structured in a table format.
- When your application requires a large number of modest continuous reads and writes and all you require is quick in-memory access.
- When your application requires CRUD (Create, Read, Update, Delete) functionality.
- These are often adaptable and perform well when your application has to run across a broad range of access patterns and data kinds.

How does a Document Database Work?

It appears that document databases work under the assumption that any kind of information can be stored in a document. This suggests that you shouldn't have to worry about the database being unable to interpret any combination of data types. Naturally, in practice, most document databases continue to use some sort of schema with a predetermined structure and file format.

Document stores do not have the same foibles and limitations as SQL databases, which are both tubular and relational. This implies that using the information at hand is significantly simpler and running queries may also be much simpler. Ironically, you can execute the same types of operations in a document storage that you can in a SQL database, including removing, adding, and querying.

Each document requires a key of some kind, as was previously mentioned, and this key is given to it through a unique ID. This unique ID processes the document directly instead of being obtained column by column.

Document databases often have a lower level of security than SQL databases. As a result, you really need to think about database security, and utilizing Static Application Security Testing (SAST) is one approach to do so. SAST, examines the source code directly to hunt for flaws. Another option is to use DAST, a dynamic version that can aid in preventing NoSQL injections.

Document database advantages

One major benefit of document-store is that all of the data is stored in a single location, rather than being spread out over many interconnected databases. As a result, if you do not employ relational processes, you perform better than a SQL database.

- **Schema-less:** Because there are no constraints on the format and structure of data storage, they are particularly effective at keeping huge quantities of existing data.
- **Faster creation of document and maintenance:** The creation of a document is a fairly straightforward process, and apart from that, the upkeep requirements are virtually nonexistent.
- **Open formats:** It offers a relatively easy construction process that makes use of XML, JSON, and other formats.
- **Built-in versioning:** Because it contains built-in versioning, it means that when the documents expand in size, there is a possibility that they will also expand in complexity. Versioning makes conflicts less likely.

More precisely, document stores are excellent for the following applications because schema can be changed without any downtime or because you could not know future user needs:

- eCommerce giants (Like Amazon)
- Blogging platforms (such as Blogger, Tumblr)
- CMS (Content management systems) (Like WordPress, windows registry)
- Analytical platforms (such as Tableau, Oracle server)

Document databases' drawbacks

- **Weak Atomicity:** Multi-document ACID transactions are not supported. We will need to perform two different queries, one for each collection, in order to handle a change in the document data model involving two collections. This is where the atomicity criteria are violated.
- **Consistency Check Limitations:** A database performance issue may arise from searching for documents and collections that aren't linked to an author collection.
- **Security:** In today's world, many online apps do not have enough security, which in turn leads to the disclosure of critical data. Thus, web app vulnerabilities become a cause for concern.

Document databases examples

- One of the best NoSQL database engines is **MongoDB**, which is not only well-known but also uses JSON like format. It has its own query language.
- A search engine built on the document-store data architecture is **Elasticsearch**. Database searching and indexing may be accomplished using this straightforward and easy-to-learn tool.
- **CouchDB:** In addition to Ubuntu, it also works with the social networking site Facebook. It utilizes Javascript and is developed in the Erlang programming language.
- **BaseX** is a simple, open-source, XML-based DBM that makes use of Java.

Check Your Progress 2

1) How Does a Column Database Work? Discuss.

.....
.....
.....

2) What are the different Graph Database Examples?

.....
.....
.....

3) Explain document based NoSQL database.

.....
.....
.....

15.4 SUMMARY

This unit covered the fundamentals of NoSQL as well as the many kinds of NoSQL databases, such as those based on columns, graphs, key-value pairs, and

documents. Numerous businesses now use NoSQL. It is difficult to pick the best database platform. NoSQL databases are used by many businesses because of their ability to handle mission-critical applications while decreasing risk, data spread, and total cost of ownership.

Despite their incredible capability, column-store databases do have their own set of problems. Due to the fact that columns require numerous writes to the disk, for instance, the way the data is written results in a certain lack of consistency. Graph databases can be used to offer content in high-performance scenarios while producing threads that are simple to comprehend for the typical user, beyond merely expressive information in a graphical and effective way (such as in the case of Twitter). The simplicity of a key-value store is what makes it so brilliant. Although this has potential drawbacks, particularly when dealing with more complicated issues like financial transactions, it was designed specifically to fill in relational databases' inadequacies. We may create a pipeline that is even more effective by combining relational and non-relational technologies, whether we are working with users or data analysis. Document-store data models are quite popular and regularly used due to their versatility. It helps analytics by making it easy for firms to store multiple sorts of data for later use.

15.5 SOLUTIONS/ANSWERS

Check Your Progress 1

- 1) NoSQL is a way to build databases that can accommodate many different kinds of information, such as key-value pairs, multimedia files, documents, columnar data, graphs, external files, and more. In order to facilitate the development of cutting-edge applications, NoSQL was designed to work with a variety of different data models and schemas.
- 2)
 - Schema flexibility
 - Horizontal scaling
 - Quick responses to queries as a result of the data model
 - Ease of use for software developers
- 3) It is different in the following ways:
 - For the most part, NoSQL databases fall under the category of non-relational or distributed databases, while SQL databases are classified as Relational Database Management Systems (RDBMS).
 - Databases that use the Structured Query Language (SQL) are table-oriented, while NoSQL databases use either document-oriented or key-value pairs or wide-column stores, or graph databases.
 - Unlike NoSQL databases, which have dynamic or flexible schema to manage unstructured data, SQL databases have a strict, preset or static schema.
 - Structured data is stored using SQL, whereas both structured and unstructured data can be stored using NoSQL.
 - SQL databases are thought to be scalable in a vertical direction, whereas NoSQL databases are thought to be scalable in a horizontal direction.

- Increasing the computing capability of your hardware is the first step in the scaling process for SQL databases. In contrast, NoSQL databases scale by distributing the load over multiple servers.
- MySQL, SQLite, Oracle SQL, PostgreSQL, and Microsoft SQL Server are all examples of SQL databases. BigTable, MongoDB, Redis, Cassandra, RavenDb, Hbase, CouchDB, and Neo4j are a few examples of NoSQL databases.

Check Your Progress 2

- 1) A columnar database is a type of database management system (DBMS) that allows data to be stored in columns rather than rows. It is accountable for reducing the amount of time needed to return a certain query. Additionally, it is accountable for the significant enhancement of the disk I/O performance. Both data analytics and data warehousing benefit from it. Additionally, the primary goal of a Columnar Database is to read and write data in an efficient manner. Column-store databases include Casandra, CosmoDB, Bigtable, and HBase, to name a few. Also, refer 15.3.1.

- 2) Graph Database Examples:

- **Neo4j** is both an open-source and an interestingly developed on Java graph database. It is considered to be one of the best graph databases in the world. In addition to that, it comes with its own language known as Cypher, which is comparable to the declarative SQL language but is designed to work with graphs. In addition to Java, it supports a number of other popular programming languages, including Python, .NET, JavaScript, and a few others. Neo4j excels in applications such as the administration of data centres and the identification of fraudulent activity.
- **RedisGraph** is a graph module that is integrated into Redis, which is a key-value NoSQL database. RedisGraph was developed to have its data saved in RAM for the same reason that Redis itself is constructed on in-memory data structures. As a result, a graph database with excellent speed and quick searching and indexing is created. RedisGraph also makes use of Cypher, which is ideal if you're a programmer or data scientist looking for greater database flexibility. Applications that require blazing-fast performance are the main uses.
- **OrientDB:** It's interesting to note that OrientDB supports graph, document store, key-value store, and object-based data formats. Having stated that, the graph model, which uses direct links between databases, is used to hold all of the relationships.

- 3) It is generally agreed that document stores, which are a sort of NoSQL database, are the most advanced of the available options. They use JSON as their data storage format, which is different from the more traditional rows and columns layout. Most of the day-to-day activities that we carry out on the internet are supported by NoSQL databases. NoSQL is everywhere nowadays; just look at Twitter and its use of FlockDB or Amazon and their use of DynamoDB. Also, refer 15.3.4.

15.6 FURTHER READINGS

- 1) Next Generation Databases: NoSQL and Big Data 1st ed. Edition, G. Harrison, Apress, December 26, 2015.
- 2) Shashank Tiwari, Professional NoSQL, 1st Edition, Wrox, September 2011.
- 3) <https://www.kdnuggets.com/>



UNIT 16 EMERGING DATABASE MODELS

Structure	Page no.
16.0 Introduction	
16.1 Objectives	
16.2 Distributed Databases	
16.2.1 Data Fragmentation and Replication	
16.2.2 Distributed Query Processing	
16.3 Active Databases	
16.4 XML for Data Representation	
16.5 Blockchain Databases	
16.6 Multimedia Database	
16.7 Use of Databases in Web Applications	
16.8 Summary	
16.9 Solutions / Answers	

16.0 INTRODUCTION

With the advent of relational database systems in the 1970s, database technology became popular in the industry due to the simplicity of database technologies and the availability of SQL for querying the database. However, just a relational model was not sufficient. Many advanced database technologies have become available. We have already discussed some of these technologies, like object-oriented database management systems, data warehousing and mining and NoSQL databases, in the first three Units of this Block. This Unit discusses several advanced database technologies.

This Unit first introduces you to the distributed database systems needed to address the needs of organisations that have distributed data. A distributed database system allows the distribution of fragments of data over a number of database sites. It also supports query processing, which may involve several sites. These concepts are detailed in this Unit. The Unit also introduces you to the concepts of Active databases, XML data and Blockchain technology. Finally, the Unit defines how a database system can be used as a backend to a web application.

This Unit gives a brief introduction to these database technologies. You may refer to the further readings for more details on these technologies.

16.1 OBJECTIVES

After going through this Unit, you should be able to:

- define the need for a distributed database system;
- explain the data fragmentation and replication in distributed databases;
- define the distributed query processing;
- define the features of the active database management system;
- explain the document creation using XML;
- explain the characteristics of Blockchain systems;
- list the characteristics of multimedia database
- use the web database in a web application.

16.2 DISTRIBUTED DATABASES

Many commercial organisations use a database system to manage large amounts of transactional data. These database systems have multiple users and run on a high-performance centralised computer system. These systems may allow geographically distributed users to connect to the database using a network. However, with the increase in the volume of database transactions and user interactions, use of a number of database servers, which may process local data, may be more efficient. This led to creation of distributed database management systems. A distributed database management system (DDBMS) manages several database servers, which may be dispersed at various geographical locations but store the data of a single database system. For example, consider a sample student relational schema given in Figure 1; how can this schema be represented in a distributed database? Figure 2 shows an example of a distribution of the student database over a few possible locations.

Student									
Enrollment No.	Name	Father's Name	Highest Qualification	Email	Phone	Regional Centre	Programme		
Fee									
Enrollment No.		Semester		Amount Paid		Date of Payment			
Subject									
Enrollment No.		Semester		Subject Code					

Figure 1: A sample student relation

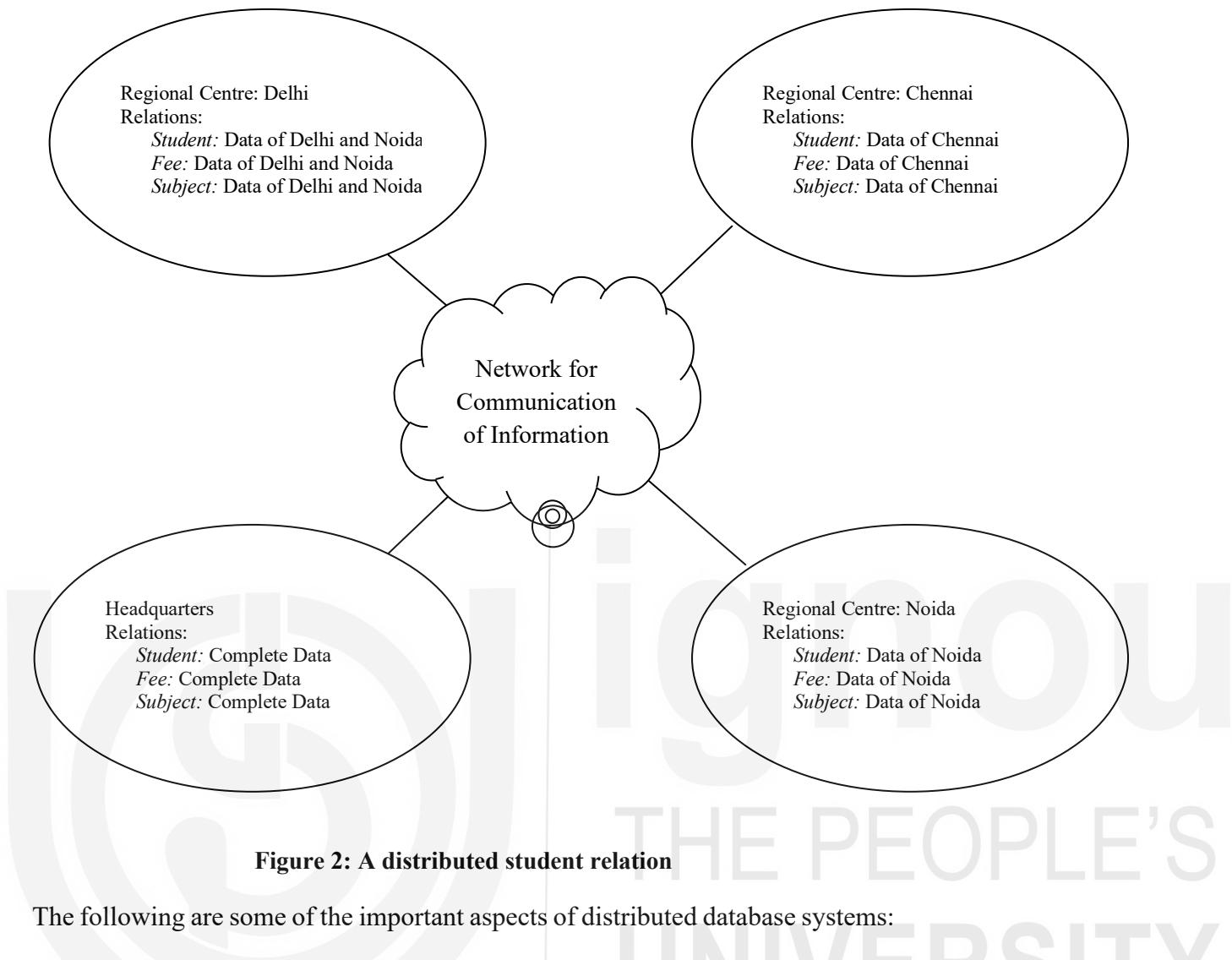


Figure 2: A distributed student relation

The following are some of the important aspects of distributed database systems:

- A distributed database has multiple sites that hold a part of a logically connected database. For example, in Figure 2, the Regional Centre Delhi holds the data of the students who are registered with that regional office of the head office.
- The technical implementation details, such as "The data of Delhi students is available at the Headquarters and Delhi Regional Center", are hidden from the actual users of the database, who can be the students or Regional Centre staff, etc. This is known as transparency.
- The user can issue a command to the DDBMS without worrying about the location of data (called location transparency), where the copies of the data have been kept (called replication transparency), or the fragment of the horizontal fragment of data is kept (called sharding). For example, in Figure 2, a student who is associated with Regional Centre Noida can place his/her query without knowing the fact that his/her data is located at the Headquarters, Regional Centre Delhi and Regional Centre Noida. His/her query will be replied to by any of the locations based on the place of the query and the status of different sites. You may also observe that in Figure 2, the data of students related to Regional Centre Noida is replicated at three sites – Headquarters, Regional Centre Delhi and Regional Centre Noida. Also, note that horizontal fragments of student

- data of Chennai are stored at the Regional Centre Chennai site.
- A DDBMS will be required to manage more failures than a centralised DBMS, as it must also manage network failures.
 - A DDBMS is available for a longer duration than that of centralised RDBMS.
 - A DDBMS is better scalable than a centralised RDBMS due to data distribution on various sites.

16.2.1 Data Fragmentation and Replication

In a distributed database, as shown in Figure 2, all the data is not stored at all the database sites. In general, the data related to a particular site is stored on that site. For example, in Figure 2, data related to Regional Centre Chennai and Regional Centre Noida is stored at their respective sites. The process of distributing data into different parts is called fragmentation. This kind of distribution of data facilitates faster query processing, as most of the queries at a site can be answered from the local data. In addition to fragmentation, the data is replicated at more than one site. For example, in Figure 2, data of Regional Centre Noida is replicated at Regional Centre Noida and Regional Centre Delhi sites. This data replication helps improve the reliability and availability of the database, as the database would be available to users even if one of the replicated sites fails. The following example explains different kinds of fragmentation.

Example: Consider the slightly modified Student table given in Figure 1 with the following database instance:

Student						
EnrNo	Name	Father	Qual	Phone	RC	Programme
23001	Anil	Mohan	UG	9900100000	Noida	PGDCA
23002	Rahim	Jamil	UG	9900200000	Chennai	PGDCA
23003	Simon	Robert	PG	9900300000	Noida	MCA
23004	Sahil	Sanjay	UG	9900400000	Delhi	MCA
23005	Sanjay	Ajay	PG	9900300000	Noida	PGDCA
23006	Diya	Jeba	UG	9900400000	Chennai	MCA

The following can be the Horizontal Fragments of the table based on the RC.

Student: Horizontal Fragment on RC Noida site						
EnrNo	Name	Father	Qual	Phone	RC	Programme
23001	Anil	Mohan	UG	9900100000	Noida	PGDCA
23003	Simon	Robert	PG	9900300000	Noida	MCA
23005	Sanjay	Ajay	PG	9900300000	Noida	PGDCA

Student: Horizontal Fragment on RC Chennai site						
EnrNo	Name	Father	Qual	Phone	RC	Programme
23002	Rahim	Jamil	UG	9900200000	Chennai	PGDCA
23006	Diya	Jeba	UG	9900400000	Chennai	MCA

Student: Horizontal Fragment on RC Delhi site (Noida or Delhi)						
EnrNo	Name	Father	Qual	Phone	RC	Programme
23001	Anil	Mohan	UG	9900100000	Noida	PGDCA
23003	Simon	Robert	PG	9900300000	Noida	MCA
23004	Sahil	Sanjay	UG	9900400000	Delhi	MCA
23005	Sanjay	Ajay	PG	9900300000	Noida	PGDCA

The other kind of fragmentation is vertical fragmentation. For example, if RC Noida has been assigned the work of contacting all the students telephonically, irrespective of their regional centre, then one possible vertical fragment for Regional Centre Noida would be as follows:

Student: Vertical Fragmentation for Noida			
EnrNo	Name	Phone	Programme
23001	Anil	9900100000	PGDCA
23002	Rahim	9900200000	PGDCA
23003	Simon	9900300000	MCA
23004	Sahil	9900400000	MCA
23005	Sanjay	9900300000	PGDCA
23006	Diya	9900400000	MCA

Further, considering that Regional Centre Noida is assigned PGDCA students only for telephonic contact, the fragment would be a mixed fragment as follows:

Student: Mixed Fragmentation for Noida			
EnrNo	Name	Phone	Programme
23001	Anil	9900100000	PGDCA
23002	Rahim	9900200000	PGDCA
23005	Sanjay	9900300000	PGDCA

As far as replication is concerned, you may observe that the student data of Regional Centre Noida is replicated at Regional Centre Delhi, too. Replication helps in enhancing reliability and availability but results in more overheads in transaction processing.

16.2.2 Distributed Query Processing

A query in a distributed database management system is submitted at a site. This query is then converted to a relational algebraic query and optimised using local and global query optimisation processes. Local query optimisation is the same as that of a centralised DBMS; however, global query optimisation involves the selection of sites for query evaluation, cost of data communication, and cost of query processing. The process of distributed query processing is explained with the help of the following example.

Example: Consider a query submitted at the Headquarters seeking to find the Percentage of fee share of Regional Centre Noida in the financial year 2022-23. This query would require computing the fee collected by RC Noida to the total fee collected between the dates 01st April 2022 and 31st March 2023. This query may consist of two subqueries:

- (a) Finding the total fee collected for the financial year 2022-23.
- (b) Finding the total fee collected at RC Noida in the financial year 2022-23.

In addition, you may assume that the financial records are ordered chronologically.

One of the possible ways of processing the queries would be to process both the sub-queries at the Headquarters. If both these sub-queries can be processed during the same query processing cycle, then it may be a good choice. However, if both the sub-queries are processed separately, then other options may be explored. What if subquery (a) is processed at Headquarters and subquery (b) is processed at the RC Noida site? This will require a transfer of results from subquery (b) from the RC Noida site to the headquarters site, where the result will be displayed to the user who made the query.

A detailed discussion on distributed database management systems is beyond the scope of this Unit. You may refer to the further readings for more details on this topic.

16.3 ACTIVE DATABASES

Active databases, as the name suggests, comprise dynamic actions on the occurrence of certain events. Such actions were part of the SQL 99 standard and are called triggers. Let us define the model that can be used for an active database:

Active Database Model

Consider the Student and Result relations given in Figure 3.

Student				
Enrollment No.	Name	Programme	Cumulative Grade Point Average	Status

Result		
Enrollment No.	CourseCode	Grade

Figure 3: An example

Assuming that the Cumulative Grade Point Average is to be updated for each student when related data is created in the Result relation. The following events will cause a database action to be activated:

Action Triggering Event: In the database of Figure 3, on updating the Result table, the Cumulative Grade Point Average (CGPA) of a Student needs to be updated, as well. Assuming that once a Record is entered in the Result table, then it cannot be deleted, and only the Grade attribute can be modified in the Result relation, the following may be the events that may trigger the action of an update on CGPA for the Student relation:

- Addition of a tuple in the Result relation
- Modification of a Grade in the Result Relation.

Once the trigger event occurs, the next step is to check the condition, if any. In the case of the Student database given in Figure 3, the referential integrity constraints on the Enrollment number and CourseCode (each student will register for a set of courses) will make sure that only the valid entry is made in the Result table. In addition, the check constraint on Grade ensures that a valid Grade letter is entered in the Result table. However, an interesting observation here is that the state of a student must be "Active" in case his/her CGPA is to be updated. This can fit as a condition for activating the trigger.

Issues relating to Active databases: Some of the issues relating to active databases relate primarily to the process of creating and maintaining the triggers. Such databases must include a set of rules or commands that should be able to deactivate or drop the old or redundant rules, as there can be a very large number of rules which may be changed during the lifetime of the database. Next, since triggers can be activated either before, during or after a condition, a huge number of events may be scheduled, causing the database to act slowly as a response to normal record addition and deletion processes. Interestingly, it adds another issue, considering that addition of results by a teacher in the student's database triggered the update of about 50 students' GPA. However, the teacher realised that s/he had used an incorrect file to update student records and deleted his/her result. This will lead to the firing of a very large number of triggers that would be required to undo the changes in the GPA of the students.

A detailed discussion on active databases is beyond the scope of this Unit. You may refer to further readings for more details.

☛ Check Your Progress 1

- 1) How is DDBMS different from RDBMS?

.....
.....

- 2) Suggest the data fragmentation of the student table if all RCs need only the enrolment number and name of all the PGDCA students.
-
.....
.....

- 3) What is an active database? What is a triggering event?
-
.....
.....

16.4 XML AND DATA REPRESENTATION

The eXtensible Markup Language (XML) is one of the popular data representation languages. It uses user-defined tags to represent a document. A typical XML document relating to the student's table, as shown in Figure 3, is shown below:

```
<school>
  <class>
    <class_no>XII</class_no>
    <class_teacher>John</class_teacher>
    <student>
      <enrolmentNo>2301002297</enrolmentNo>
      <name>Ritesh Jain</name>
      <programme>PGDCA</programme>
      <cgpa>7.5</cgpa>
      <status>Pass</status>
      <result>
        <coursecode>BCS011</coursecode>
        <grade>A</grade>
        <coursecode>BCS013</coursecode>
        <grade>B+</grade>
      </result>
    </student>
    <student>
      <enrolmentNo>2301002301</enrolmentNo>
      <name>Amitesh</name>
      <programme>PGDCA</programme>
      <cgpa>8.5</cgpa>
      <status>Distinction</status>
      <result>
        <coursecode>BCS011</coursecode>
        <grade>A+</grade>
        <coursecode>BCS013</coursecode>
```

```

<grade>A</grade>
</result>
</student>
</class>
</school>

```

Figure 4: A sample XML document

You may please observe that instead of using a separate table, in the XML document, the results of the students are merged along with the student information. Thus, XML representation has the potential to store all the information about an entity in one place. Such a representation, though it may be useful for searching from a point of view as no join operation is required, may lead to redundancy of information. In addition to the use of tags, XML allows users to store attributes along with a tag. For example, the enrolment number can be stored as an attribute of the student as:

```

<student enrolmentNo = “2301002301”>
    <name> ...
</student>
...

```

The attribute may be useful for searching for information on the related field.

Just like the database management system has a different schema, XML also can be used to validate the structure of the XML data. Figure 5 is a possible document type definition (DTD) that would validate the document given in Figure 4.

```

<!DOCTYPE school [
<!ELEMENT school (class+)>
<!ELEMENT class (class_no, class_teacher, student+)>
<!ELEMENT student (enrolmentNo, name, programme, cgpa, status, result *)>
<!ELEMENT result ( (coursecode, grade)+)>
<!ELEMENT class_no( #PCDATA )>
<!ELEMENT class_teacher( #PCDATA )>
<!ELEMENT enrolmentno( #PCDATA )>
<!ELEMENT name ( #PCDATA )>
<!ELEMENT programme ( #PCDATA )>
<!ELEMENT cgpa ( #PCDATA )>
<!ELEMENT status ( #PCDATA )>
<!ELEMENT coursecode ( #PCDATA )>
<!ELEMENT grade ( #PCDATA )>
]>

```

Figure 5: Document Type Definition for XML of Figure 4

Please note the following points in the DTD, as shown in Figure 5:

- A school consists of data from one or more classes.
- Each class stores the class number and class teacher's name.
- A class have one or more students.

- For each student, you store the enrolment number, name, programme, his/her CGPA and present status (Distinction, Merit, Pass, Unsuccessful).
- In addition, the result of each student is stored. This result can be in zero or more subjects.
- The filed type #PCDATA means parsed character data, which stores the text parsed by the parser into the fields.

XML has become a popular format for data exchange and storage. Several tools have been developed to verify, display as tree nodes and query the XML documents. A detailed discussion of these tools is beyond the scope of this Unit. However, we present a few basic features of XQuery, which is a standard query language for XML documents.

A XQuery expression uses five basic keywords for querying. These are *for*, *let*, *where*, *order by* and *return*. The *for* clause selects a sequence of nodes in a document, *let* is used to bind a variable name to a sequence, *where* is used for the selection of nodes, *order by* is used for giving an order to the sequence, and *return* is used to specify what values are to be returned.

For example, the query:

```
for $x in /school/class/student
where $x/cgpa > 8
return </name>
```

This query will return the name of the student with a CGPA of 8 or more. You may refer to the further readings for more details on XQuery.

16.5 BLOCKCHAIN DATABASES

Conceptually, blockchain is a paradigm of storage of data in a distributed manner, which may also protect data from fraudulent transactions and updates. Blockchain technology was used to store distributed ledgers and bitcoins. However, this technology is not limited to only these applications. In this section, we will present some of the basic features of blockchain with the help of an example. However, to understand the principles of blockchain, you should study the paper "Bitcoin: A Peer-to-Peer Electronic Cash System" by Satoshi Nakamoto in 2008.

Let us discuss the components of the blockchain technology. A Blockchain will have the following components:

1. Distributed set of Authorised Nodes: The role of a node is to keep the ledger of data. A ledger consists of data logs, which are timestamped. A *blockchain* is a sequence of *blocks* of data, which is maintained at each of these authorised nodes.
2. A *block* contains:
 - a. Block Number: It is a unique sequence number given to every

- block on the blockchain.
- Nonce: It is a random number that is used only once in a block.
 - Transaction logs: A sequence of transaction logs that are to be recorded in a block.
 - Computed Hash value: A cryptographic hash function is applied to the content of a block to compute a hash value of the block. This hash value is also stored in the block.
 - Hash value of Previous Block.
3. A blockchain is constructed by linking the blocks. In addition, each block contains the hash value of previous block, which is used to ensure data security.

A hypothetical blockchain is shown in Figure 6.

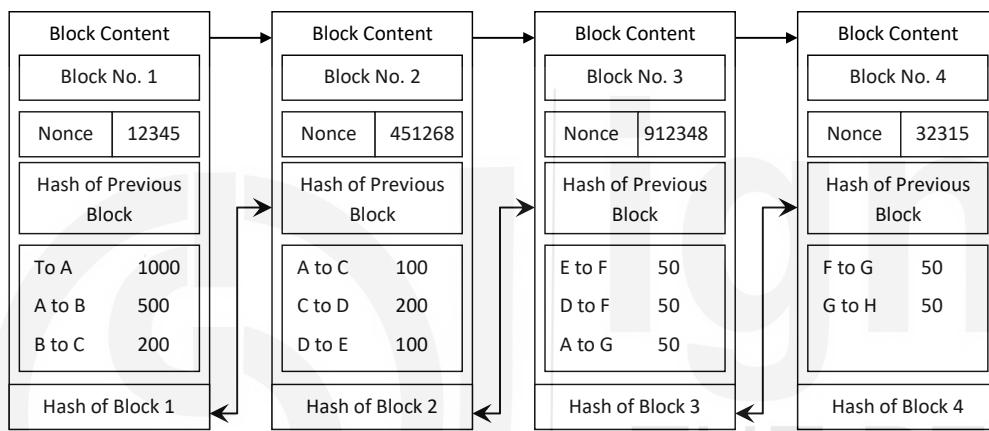


Figure 6: A Hypothetical Blockchain Data

The blockchain processes involve the following:

1. A network consensus protocol, which ensures security, trust, and concurrence amongst the network of nodes of the blockchain.
2. The hashing process ensures the integrity of the transaction ledgers; in other words, it ensures that the transaction ledgers are not tampered with. In Figure 6, each transfer of money may represent one entry of the transaction ledger.
3. Uses a digital signature to certify consent for the transaction.

Let us discuss some of these concepts in more detail.

Cryptographic Hash Function: At the most elementary level, a blockchain consists of a cryptographic Hash function. This hash function is a kind of secure fingerprint of a Block of data of blockchain. One of the popular hash algorithms is Secure Hash Function 256 (SHA256), which produces a 256-bit long hash value for the content of a block irrespective of the size of the block (which can vary from one character to millions of characters). The hash function has the following properties:

- It maps the block contents of any size to a hash value called Hash of a fixed size; for example, SHA 256 maps block content of any size to 256 bits Hash.
- For a given content, the hash function will always produce the same Hash. In addition, it is expected that if the contents of two blocks are different even by a single character, then the hash function should produce different Hash values for the blocks.
- In case you change the content of a block slightly, then the value of the Hash function changes significantly. In addition, if you know a part of the hash value, you cannot predict the rest of the hash value.
- The cryptographic hash function is also called one-way encryption, as you can use the block contents to create the Hash value, but you cannot use the Hash value to find the block contents. Interestingly, many password-based authentication systems store the password using one-way encryption. This is why your system administrator simply provides you a link to change the password, but not the password itself.

Please also note that in case there is a change in the content of a block, its hash value will also change. This feature of the hash value may be very useful in finding the tampering of data of a block, which is discussed next.

How to identify data tampering or corruption in a Blockchain?

As discussed earlier, any minor change of any value in a block at a particular node will result in change in its computed hash value. This will cause change in the stored *hash value of previous block* in all the subsequent blocks. For example, consider in Block No. 2 of Figure 6, the transaction log of A to C is modified to 200 from the present value of 100. This will result in change in the computed hash value of Block no. 2. Further, this will result in change in *hash value of previous block* of Block 3, which will result in change in the computed hash values of Block no 3. A similar change will occur in Block 4, too. Thus, the previously stored hash values of Block 2, Block 3 and Block 4 will not match the computed new hash values of these blocks.

But how will it be recognised that the blocks have different hash values? Here, the consensus protocol will play its role. Since each blockchain block is stored on all the blockchain network nodes, changes in one site can easily be recognised, as the other nodes with the stored copy of the blockchain will not agree with this blockchain, thus identifying the tampering.

Use of Nonce: Nonce is part of the block content and is a short form of random Number used once. This number is mined using an algorithm such that the Hash value generated for the block is unique.

Validity of Transactions: You may observe that in Figure 6, most of the transactions show the transfer of money from one account to another. For example, in Block 1, there is a transaction - A to C 500. But does A have that much amount? Well, that can be established from the first transaction in this block, which says "To A 1000". Thus establishing that A has sufficient funds for

the transfer. Such transactions are sometimes called Coinbase transactions and can be used to ascertain the validity of transactions.

Use of Digital Signature: The digital signatures are used to ensure that the transactions are performed by the authorised person. For example, consider a new transaction, "A to E 50", to be added to Block 4; how will it be ensured that this transaction has been performed by A? For this, A needs to have a pair of private and public keys; let us call them PrivateKeyA and PublicKeyB. Both these keys can be very long in length. A must maintain the PrivateKeyA as a secret key, whereas he can share the PublicKeyA with other stakeholders. One key characteristic of this private-public key pair is that if you know the PublicKeyA, you cannot derive the PrivatekeyA. A can create this transaction "A to E 50" by using the PublickeyA and PublickeyE as:

"PublickeyA toPublickeyE 50"

Next, A needs to sign this message using the PrivatekeyA.

This transaction can be verified by using the PublickeyA, which ensures that the transaction has been created by an authorised person only. Please note here that in Figure 6, we have shown transactions using alphabets like A, B, C, etc., but in the actual blockchain, they will be public keys.

You may refer to the further readings for more details on Blockchain technology.

☞ Check Your Progress 2

- 1) Create an XML document consisting of Marks of two students in at least one subject. Also, make the DTD for validating this XML document.

.....
.....

- 2) What is the role of nonce in a blockchain?

.....
.....

- 3) Consider that in Figure 6, in Block 3, the E to F transaction is modified to 100. What changes would it cause in the blockchain?

.....
.....

16.6 MULTIMEDIA DATABASE

Multimedia data is an integration of textual, graphical, audio, video, and animation data. In general, multimedia data may include lengthy textual documents, pictures, drawings, digital audio clips, movies, and animations. A multimedia database should be able to store large multimedia data efficiently and provide the feature of querying the multimedia data.

Querying is a very interesting domain in the context of multimedia data, as most searches in such data require retrieval of data based on some content. For example, you may be interested in all the videos related to "Database Integrity and Normalization" from a multimedia database or videos of a particular presenter. Please note that such queries would require indexing on the objects and related contents.

How can you create these indexes? One way to create indexes would be to create a large amount of metadata for each object. This metadata should use standardised keywords, title, credentials of creators, summary information, etc. However, in many situations, the metadata would not be available, leading to the generation and verification of the metadata. The second type of indexes are those that can be created using automatic analysis.

Characteristics of Multimedia Data:

Let us now discuss some of the basic characteristics of different types of multimedia data.

Textual Data: In general, the textual data consists of articles, which include paragraphs and headings. This data is stored using ASCII or Unicode standards. Further, you may define certain keywords on these articles that can be used for searching.

Image Data: An image is stored in digital form using picture elements called *pixels*. The size of the picture depends on the number of colours used. For example, a black and white picture would just require 1 bit for every pixel, whereas a true coloured picture may require 24 bits (8 bits each for three basic colours) to store one element. Further, the resolution of a picture is represented using pixels per inch. Therefore, the size of a good-quality picture is quite substantial. You can use different kinds of compression standards to reduce the size of a picture. Some of the common image files include formats like GIF, PNG, JPEG, etc. A search on images may be to find the images related to the same objects. To answer such a query, every image can be divided into segments that have similar characteristics. Further, this segment characteristics information can be grouped to identify certain basic characteristics of an image. Various images are compared based on the similarity of characteristics of those images. In addition, a number of labels, indexes, etc., can also be linked to the segments of images, helping in finding meaningful information about the images. You may have noticed that present image recognition software are able to link many images together based on their characteristics.

Video and Animation Data: Video and animation data may be considered as a large sequence of frames that are displayed in real-time to form a live movie. In general, video data may be divided into video segments, which may be related to a group of objects or activities. For example, an eLearning video may be divided into segments, with each segment involving a sequence of learning concepts. Some of the popular compression formats used for video are MPEG, AVI, etc. A query of video data may be related to identifying the portion of a video related to a specific learning objective.

Audio data: Audio data is recorded audio messages. These messages may be identified for similarity of voice. A query on audio data may try to group the audio of a person based on recognition of voice.

Issues and Characteristics of Analysis of Multimedia Data

- Multimedia data must be stored, labelled, and indexed so you can define some similarity measures on that data.
- Statistics can be used to define the characteristics of image data using the values of colour, texture, etc.
- Further, the object's shape can be one feature of object recognition. This may

include the identification of facial features.

- In the present time, the recognition of an object in an image, video or animation is a major challenge. The key barriers to object recognition are:
 - The angle from which an image has been taken may vary the shape of an object
 - The scale and size of the picture may affect the recognition of objects.
 - Rotated, transformed, and occluded images are difficult to recognise, as it is difficult to keep a database of all transformations of an image.
- An important development in the area of object recognition is the development of a scale-invariant feature transform (SIFT) by David Lowe. SIFT extracts features from images such that they are not affected by rotation and scaling. You may refer to this algorithm in further readings.
- Another important concept relating to the recognition of images is the tagging of images. Tagging is useful in searching for images. You must have observed on certain social media that you may be tagged in some images that include you. At present, the tagging algorithms are being enhanced for better accuracy. In general, these algorithms use machine learning and statistics to analyse the image content with already tagged image libraries.
- Digital audio is difficult to index and retrieve, as it does not have any typical features or characteristics that can be used to identify the content specifically.
- A multimedia database manages different data formats and, thus, requires many storage and Input/Output technologies. These technologies may include technologies for text and images, like scanners, digital cameras, printers, etc.; technologies for audio, video, and animation, like microphones, video cameras, DVDs, Musical Instrument Digital Interface (MIDI), good quality displays and speakers etc.

Multimedia data is part of many digital systems, such as Patient health monitoring data, geographical data, students' data, etc.; therefore, effective multimedia data management systems are required to handle such vast data.

16.7 USE OF DATABASES IN WEB APPLICATIONS

A database system is a persistent collection of an organisation's data, which is shared and integrated among various applications. Database technology supports non-redundant storage of data, which allows the following features:

- Structured storage of data in the form of tables
- Secure data insertion, modification, and deletion.
- Support for concurrent database transactions.
- Easy but controlled access to data.

These features are very useful for any web application, too. Therefore, many web applications use database management systems (DBMS) to store data and access it securely for display on the web. These DBMSs are normally managed on a separate server, called a database server, and communicate with the web server to store or retrieve data. The web server then communicates this information through the relevant web pages to communicate with the web clients.

For example, when you register on an eCommerce website, the information you fill in the registration form is stored in a registration database. This registration information is accessed when you log in again on that website. On the eCommerce website, you may

also search for product information, put some items in the shopping cart, order the items from the cart, and so on. All these activities are supported by several databases. However, these activities are obscure, as you just interact with the eCommerce website through a browser interface. This section focuses on how the database can be accessed by a web server; however, our approach will be to discuss the process rather than the use of a scripting language for coding.

Some of the key characteristics that should be supported by the web server–database server interconnection are:

- The information exchange between the two servers must be secure.
- The two servers should have pools of connection for transfer of information.
- Concurrent read and write operations should be supported by the database server.
- The response time for an operation should be as low as possible.

In general, these servers may either be supporting a 2-tier or a 3-tier client-server architecture.

Steps for Creating a Web Database Application

The first requirement for creating a web application is that your web server has the required drivers to connect to the DBMS of interest. You are required to perform the following steps to create a web application that uses a database as a backend.

Connecting Web Server to a Database Server

On receiving a request from a web application client, a web application may request data from a database server. In general, a web server is required to establish a connection with the database server if it has not already done so. You require the following parameters for a connection:

- A database driver for the DBMS of interest.
- Access credentials on the DBMS and the database (e.g. username and password).
- The URL of the database server and its port number.

For example, assume that you are developing a web application using Java Server Pages (JSP) and you want to connect to a MySQL database. Further, your username is *student* and password *dbms*. In addition, the MySQL host is a local host at the URL and port number - *jdbc:mysql://localhost* and 3306, respectively. Further, assume that the database which is to be accessed is named: *class_schedule* and the table of the database is *class* (*teacherName, dateofclass, timeofclass, classroom*), which have been created by you using SQL command using the username *student*. The following command would be needed by you to create an active connection:

```
Connection connection = DriverManager.getConnection  
("jdbc:mysql://localhost:3306/class_schedule", "student", "dbms");
```

You may have to perform several other commands, too. You may refer to further readings for more details.

Creating a form for data input and inserting data into the database

You may need to create a form using HTML or any scripting language and use GET or POST methods to transfer the data to the web server, which may execute the commands for inserting data into the database (only after establishing a connection with the database).

For example, the following commands will help you insert the data into the database:

```
String sqlinsert = "insert into class_schedule values (?, ?, ?, ?);  
PreparedStatement preparedstat = connection.prepareStatement(sqlinsert);  
preparedstat.setString(1, teacherName);  
preparedstat.setString(2, dateofclass);  
preparedstat.setString(3, timeofclass);  
preparedstat.setString(4, classroom);
```

```
prep.executeUpdate();
preparedstat.close();
```

The commands shown above are just to demonstrate the programming. Please note that the variable name *connection* is the same as the variable name used while establishing the connection. The insert statement consists of four '?', filled up by the data obtained from the HTML form using the *setString* functions.

Accessing data from the database and displaying it as a webpage

For this purpose, you may create a query to access the database (after establishing a connection). The data obtained from the query may be put in a variable, which can be accessed record by record or complete data at a time to create the required webpage.

For example, you may display the content of all the classes using the following set of commands. This command may result in a number of rows of information being created by the output *System.out.println* function call.

```
String sqlselect = "select * from class_schedule ";
Statement statement = connection.createStatement();
ResultSet results=statement.executeQuery (sqlselect);
while(results.next())
    System.out.println(results.getString(1)+" "+rs.getString(2)+" "+
    rs.getString(3)+" "+rs.getString(4));
```

Many more statements are required to implement a web database. You may refer to the documentation of the web tools, languages and databases you use to implement a web application. While using the connection, the term *jdbc* was used. Let us discuss such terms in more detail.

Open Database Connectivity (ODBC) and JAVA Database Connectivity (JDBC)

There are many commercial DBMSs like DB2 (IBM), Oracle and MySQL (Oracle), SQL Server, MS-Access (Microsoft), PostgreSQL (Open Source) etc. Each DBMS has its own interface for programming (called Application Programming Interface or, in short, API) and data storage and indexing mechanisms. ODBC is a standard interface (API) that allows you to connect to a database of any DBMS and manipulate data using SQL commands. If you are using JAVA to write the interface functions to a DBMS, then you require JDBC. These interfaces provide a set of drivers that allow you to connect and access data from the database.

Closing of database connection: In general, it is recommended to close the connection after the required database operation has been performed.

Check Your Progress 3

- 1) What are the characteristics of multimedia data?

.....
.....

- 2) List four application areas of multimedia databases.

.....
.....
.....

- 3) What are the steps required to access a database from a web application?

.....
.....
.....

4. Why is there a need for ODBC? Why do you need JDBC?

.....
.....
.....

16.6 SUMMARY

This Unit introduced you to some of the advanced technologies. The unit discusses the concepts relating to the distributed database management systems (DDBMS). A DDBMS replicates the fragmented data on various database sites. This unit explains the concepts of horizontal and vertical fragmentation of data. A query in distributed database may be submitted at any site of the database. This unit introduces how a distributed query will be processed. The unit also explains the triggering events and dynamic actions in the context of active database with the help of an example. XML is one of the popular ways of representing semi-structure data in a very simple format. This unit explained the data representation and validation in XML. The unit also presented a brief introduction to blockchain technology. The focus in the unit was to introduce you to basic concepts with the help of an example. Finally, the unit presents a brief introduction to multimedia database and web databases.

16.7 SOLUTIONS / ANSWERS

Check Your Progress 1

- 1) A DDBMS allows transparency to a user about the location of the data, which may be distributed among various sites of the database. DDBMS makes sure that data is made available to the user (if she has the access rights to use the data) even if one site is down. This is possible because DDBMS keeps track of the data replications. In addition, DDBMS is responsible for concurrent transaction management in distributed settings. The RDBMS, on the other hand, does not distribute data and, therefore, is less complex than DDBMS.

- 2) In such a case, you may only require the following fragment:
SELECT EnrNo, Name
FROM Student
WHERE Programme = 'PGDCA'

- 3) An active database is a database that has a set of active actions that are to be performed upon the occurrence of an event. Some of these triggering events can be the addition or deletion of a record or modification of a record.

Check Your Progress 2

- 1) The following is the XML document:

```
<studentMarks>
  <student>
    <rollno>12345</rollno>
    <name>Aman</name>
    <marksinsubject>
      <subject>Mathematics</subject>
      <marks>75</marks>
      <subject>Physics</subject>
      <marks>85</marks>
    </marksinsubject>
  </student>
  <student>
    <rollno>54321</rollno>
    <name>Arvin</name>
    <marksinsubject>
      <subject>Mathematics</subject>
      <marks>65</marks>
    </marksinsubject>
  </student>
</studentMarks>
```

The DTD would be as follows:

```
<!DOCTYPE studentMarks [
  <!ELEMENT (student (rollno, name, marksinsubject+))+>
  <!ELEMENT marksinsubject ( (subject, marks)+)>
  <!ELEMENT rollno ( #PCDATA )>
  <!ELEMENT name ( #PCDATA )>
  <!ELEMENT subject ( #PCDATA )>
  <!ELEMENT marks ( #PCDATA )>
]>
```

- 2) The nonce means the number used once. It is a random number appended to the blockchain content to ensure that no two blocks have the same hash value.
- 3) The change in the Block 3 E to F transaction to 100 will change the hash value of Block 3; this will no longer match the hash value of the previous block stored in Block 4. Further, the consensus protocol will not accept values from this block for any purpose.

Check Your Progress 3

- 1) The following are the basic characteristics of multimedia data:

Textual Data: includes long strings such as textual articles, can include multilingual UNICODE formats, and requires a keyword-based search.

Image Data: large data requiring compression, use formats like GIF, PNG, JPEG, etc. images can be segmented and tagged for the purpose of similarity

Video and Animation Data: has a large sequence of frames, video segments may be created, and popular compression formats are MPEG, AVI, etc.

Audio data: may be identified for similarity of voice.

- 2) Medical databases

Bioinformatics

3) The following steps are required to create a web database:

- You should have a valid database with proper records on a database server.
 - Create a connection string using the URL and port number of the database server. This string should also include a valid username and password.
 - Open the connection, query the database using the parameters submitted by the user, and generate the results.
 - Format and display the results on the client, which may be a browser window.
4. ODBC helps in accessing the contents of the database using SQL commands, simplifying the programming of the databases. JDBC is an application programming interface that allows Java programmers to access any database.

